Angular JS LAB Manual 1 - dfx

COMPUTER SCIENCE ENGINEERING (Jawaharlal Nehru Technological University, Kakinada)

1.A) Course Name: Angular JS
Module Name: Angular Application Setup
Observe the link http://localhost:4200/welcome on which the mCart application is running. Perform the below activities to understand the features of the application.

Angular is one of the most powerful and performance-efficient JavaScript frameworks to build single-page applications for both web and mobile. The powerful features of Angular allow us to create complex, customizable, modern, responsive, and user-friendly web applications.

Angular follows a component-oriented application design pattern to develop completely reusable and modularized web applications. Popular web platforms like Google Adwords, Google Fiber, Adsense have built their user interfaces using Angular.

In this course, you will learn about components, modules, directives, data binding, pipes, HttpClient, Routing, and much more. You will be using Angular CLI to speed up the development process of Angular applications. Angular CLI is a command-line interface tool to scaffold and build Angular applications. Angular CLI offers all best practices right from development till the deployment stage.

Angular 1 is a JavaScript framework from Google which was used for the development of web applications.

Following are the reasons to migrate from Angular 1 to the latest version of Angular:

Cross-Browser Compliant

Internet has evolved significantly from the time Angular 1.x was designed. Creating a web application that is cross-browser compliant was difficult with Angular 1.x framework. Developers had to come up with various workarounds to overcome the issues. Angular helps to create cross-browser compliant applications easily.

Typescript Support

Angular is written in Typescript and allows the user to build applications using Typescript. Typescript is a superset of JavaScript and more powerful language. The use of Typescript in application development improves productivity significantly.

Web Components Support

Component-based development is pretty much the future of web development. Angular is focused on component-based development. The use of components helps in creating loosely coupled units of application that can be developed, maintained, and tested easily.

Better support for Mobile App Development

Desktop and mobile applications have separate concerns and addressing these concerns using a single framework becomes a challenge. Angular 1 had to address the concerns of a mobile application using additional plugins. However, the Angular framework, addresses the concerns of both mobile as well as desktop applications.

Better performance

The Angular framework is better in its performance in terms of browser rendering, animation, and accessibility across all the components. This is due to the modern approach of handling issues compared to earlier Angular version 1.x.

A Single Page Application (SPA) is a web application that interacts with the user by dynamically redrawing any part of the UI without requesting an entire new page from the server.

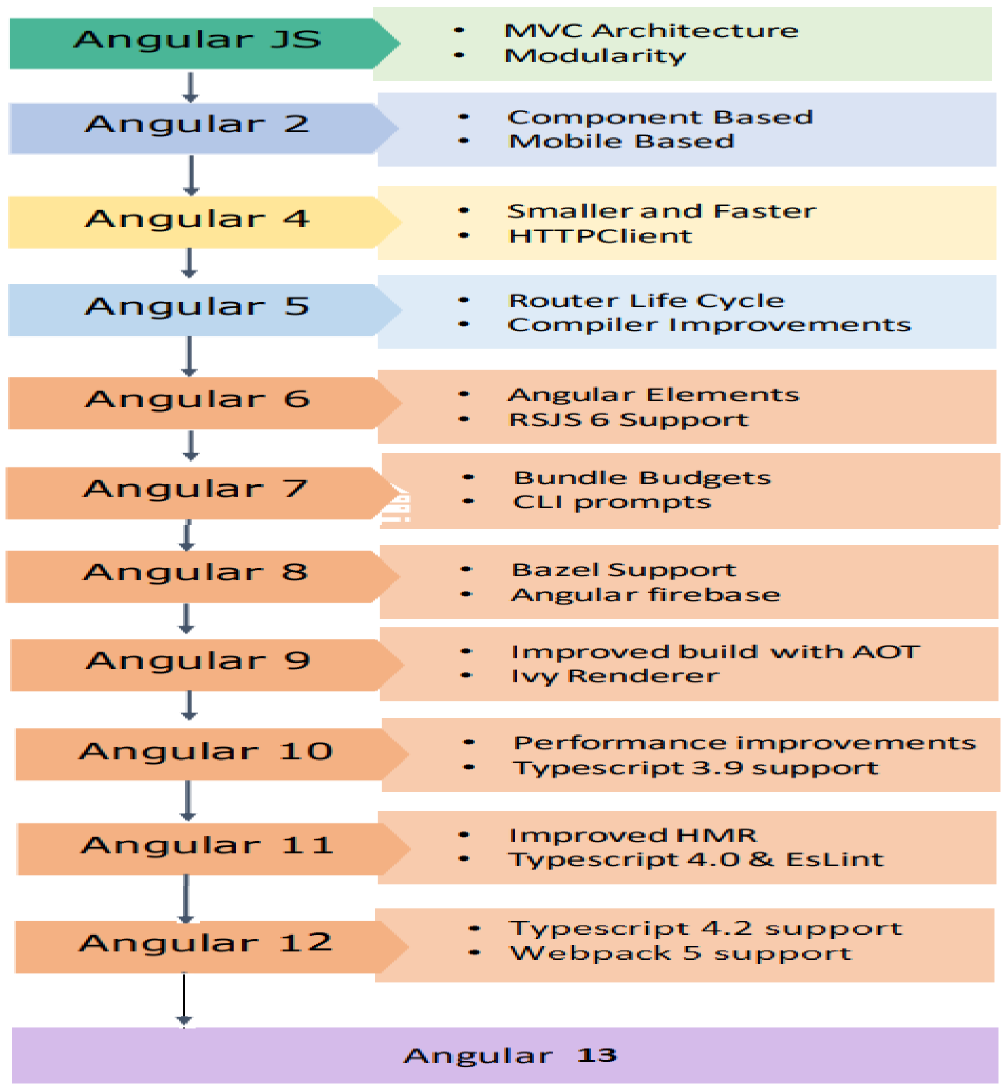
For example, have a look at the Amazon web application. When you click on the various links present in the navbar present in any of the web pages of this application, the whole page gets refreshed. This is because visibly, a new request is sent for the new page for almost each user click. You may hence observe that it is not a SPA.

But, if you look at the Gmail web application, you will observe that all user interactions are being handled without completely refreshing the page.

Modern web applications are generally SPAs. SPAs provide a good user experience by communicating asynchronously (a preferable way of communication) with a remote web server (generally using HTTP protocol) to dynamically check the user inputs or interactions and give constant feedback to the user in case of any errors, or wrongful/invalid user interaction. They are built block-by-block making all the functionalities independent of each other. All desktop apps are SPAs in the sense that only the required area gets changed based on user requests.

Angular helps to create SPAs that will dynamically load contents in a single HTML file, giving the user an illusion that the application is just a single page.

Evolution of Angular Framework:



Let us now understand what is Angular and what kind of applications can be built using Angular:

- Angular is an open-source **JavaScript** framework for building both mobile and desktop web applications.
- Angular is exclusively used to build **Single Page Applications (SPA).**
- Angular is completely rewritten and is not an upgrade to Angular 1.
- Developers prefer TypeScript to write Angular code. But other than TypeScript, you can also write code using JavaScript (ES5 or ECMAScript 5).

Why most developers prefer TypeScript for Angular?

- TypeScript is Microsoft's extension for JavaScript which supports object-oriented features and has a strong typing system that enhances productivity.
- TypeScript supports many features like annotations, decorators, generics, etc. A very good number of IDE's like Sublime Text, Visual Studio Code, Nodeclipse, etc., are available with TypeScript support.
- TypeScript code is compiled to JavaScript code using build tools like npm, bower, gulp, webpack, etc., to make the browser understand the code.
- In this course, you will learn the Angular framework by exploring the implementations of the business requirements of an application called **mCart**.
-
- mCart is an **online shopping application** that helps the users to purchase mobiles and tablet devices. This application allows users to log in for purchasing mobiles and tablet devices. They can select the products and add them to the cart. Once the selection is done, they can go to the cart page for payment. They can search for a product, sort the products list based on rating or price, and can filter the products list based on the manufacturer, operating system, and price.
-

User Stories
Login to the application to buy tablets/mobiles
Search for a specific product
Filter products based on manufacturer, price, and operating system
View the details of a specific product
Sort the products based on popularity and price
Add products to the cart which you want to buy
Change the quantity of the products selected for purchase
Checkout for closing the purchase
Log out from the application

To develop an application using Angular on a local system, you need to set up a development environment that includes the installation of:

- Node.js (^12.20.2 || ^14.15.5 || ^16.10.0) and npm (min version required 6.13.4)
- Angular CLI
- Visual Studio Code

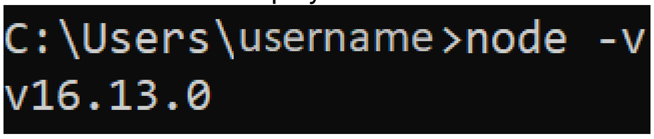
1. Steps to install Node.js

Install Node.js (^12.20.2 || ^14.15.5 || ^16.10.0) from Sparsh Downloads as shown below and take help from CCD to get it installed.

To check whether the Node is installed or not in your machine, go to the Node command prompt and check the Node version by typing the following command.

```
node -v
```

It will display the version of the node installed.



```
C:\Users\username>node -v
v16.13.0
```

2. Steps to install Angular CLI

Angular CLI can be installed using node package manager as shown below:

Note: Node modules should be downloaded from Node Repository managed by Infosys. By default, Node Package Manager(NPM) points to global Node registry and downloads the Node modules from there. To make NPM point to Node Repository managed by Infosys, use below commands.

npm config set registry https://infyartifactory.ad.infosys.com/artifactory/api/npm/npm

npm login

Post npm login, it will ask for username, password, and email. Enter the username without @infosys.com, enter your Infosys password and email id.

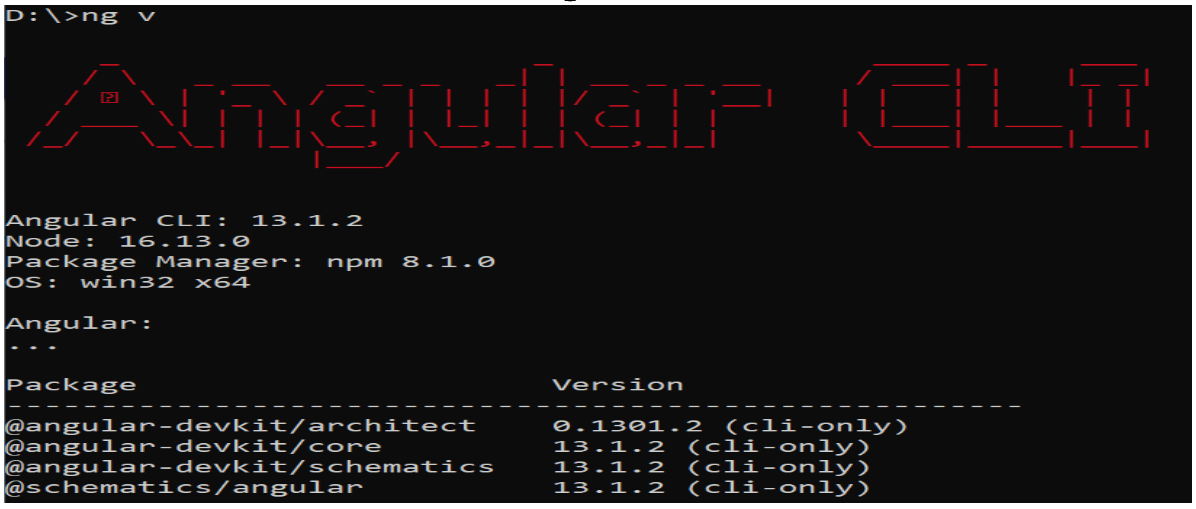
Now run the following command to install CLI.

```
D:\>npm install -g @angular/cli
```

Test successful installation of Angular CLI using the following command

Note: Sometimes additional dependencies might throw an error during CLI installation but still check whether CLI is installed or not using the following command. If the version gets displayed, you can ignore the errors.

D:\> ng v



```
D:\>ng v

Angular CLI: 13.1.2
Node: 16.13.0
Package Manager: npm 8.1.0
OS: win32 x64

Angular:
...

Package                          Version
-----------------------------------------------------------
@angular-devkit/architect        0.1301.2 (cli-only)
@angular-devkit/core             13.1.2 (cli-only)
@angular-devkit/schematics       13.1.2 (cli-only)
@schematics/angular              13.1.2 (cli-only)
```

Angular CLI is a command-line interface tool to build Angular applications. It makes application development faster and easier to maintain.

Using CLI, you can create projects, add files to them, and perform development tasks such as testing, bundling, and deployment of applications.

Command	**Purpose**
npm install -g @angular/cli	Installs Angular CLI globally
ng new <project name>	Creates a new Angular application
ng serve --open	Builds and runs the application on lite-server and launches a browser
ng generate<name>	Creates a class, component, directive, interface, module, pipe, and service
ng build	Builds the application
ng update @angular/cli @angular/core	Updates Angular to latest version

3. Steps to install Visual Studio Code

Install Visual Studio code from software house/software center as shown below or take help from CCD to get it installed:

Highlights:

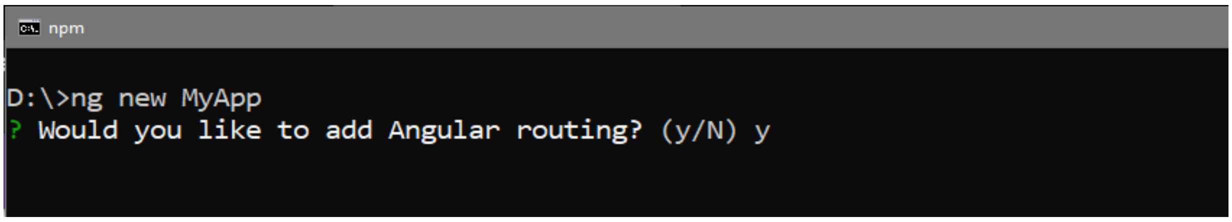
- Creating an Angular application using Angular CLI
- Exploring the Angular folder structure

Demosteps:

1. Create an application with the name 'MyApp' using the following CLI command

D:\>ng new MyApp

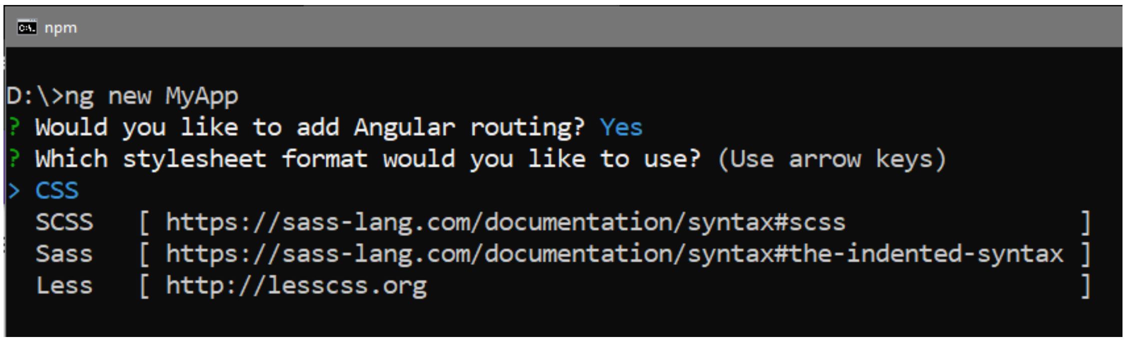
2. The above command will display two questions. The first question is as shown below. Typing 'y' will create a routing module file (app-routing.module.ts).



```
npm
D:\>ng new MyApp
? Would you like to add Angular routing? (y/N) y
```

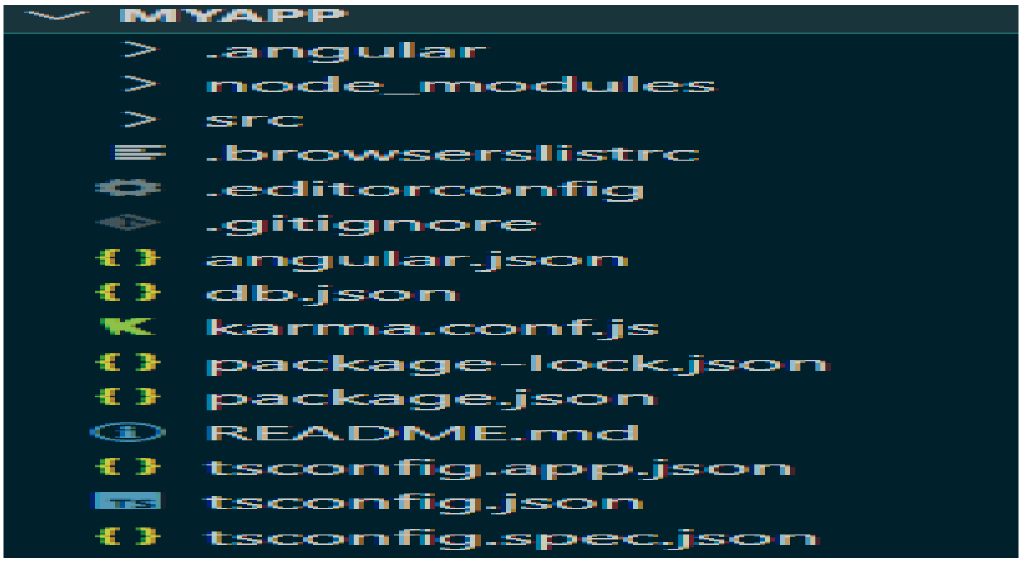
Next, you will learn about the app-routing.module.ts file.

3. The next question is to select the stylesheet to use in the application. Select CSS and press Enter as shown below:



```
npm
D:\>ng new MyApp
? Would you like to add Angular routing? Yes
? Which stylesheet format would you like to use? (Use arrow keys)
> CSS
  SCSS   [ https://sass-lang.com/documentation/syntax#scss                ]
  Sass   [ https://sass-lang.com/documentation/syntax#the-indented-syntax ]
  Less   [ http://lesscss.org                                             ]
```

This will create the following folder structure with the dependencies installed inside the node_modules folder.



```
∨ MYAPP
    > .angular
    > node_modules
    > src
      .browserslistrc
      .editorconfig
      .gitignore
      angular.json
      db.json
      karma.conf.js
      package-lock.json
      package.json
      README.md
      tsconfig.app.json
      tsconfig.json
      tsconfig.spec.json
```

Note: If the above command gives errors while installing dependencies, navigate to the project folder in the Node command prompt and run "npm install" to install the dependencies manually.

File / Folder	Purpose
node_modules/	Node.js creates this folder and puts all npm modules installed as listed in package.json
src/	All application-related files will be stored inside it
angular.json	Configuration file for Angular CLI where we set several defaults and also configure what files to be included during project build
package.json	This is a node configuration file that contains all dependencies required for Angular
tsconfig.json	This is the Typescript configuration file where we can configure compiler options
.angular	From Angular version `13.0.0, .angular` folder is generated in the root. This folder caches the builds and is ignored by git.

Download the mCart case study to your local machine. Click the following link to download the case study.

Click here to download

After downloading the project, open the **Node command prompt** and navigate to the mCart folder as shown below, and run the 'npm install' command to install the npm packages.

This will create a folder called node_modules with all the dependencies installed inside it

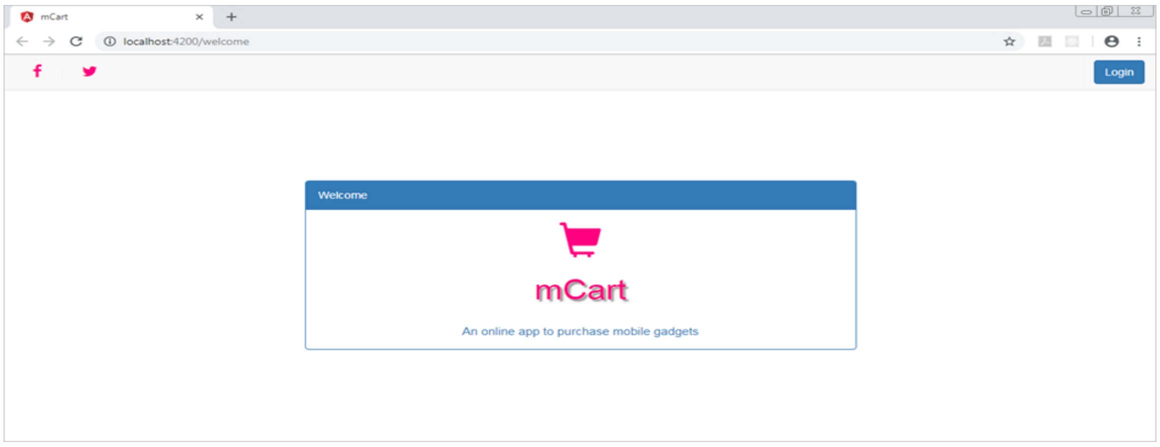
Note: Sometimes Angular will throw errors during installation. This can be due to the unavailability of some additional dependencies in your machine. After installation, always check if the node_modules folder is created under the project root folder. If it is created, you can ignore the errors that occurred during installation.

Type the following command to run the application. This will open a browser with the default port as 4200.

D:\mCart>ng serve –open

Problem Statement:

Observe the link http://localhost:4200/welcome on which the mCart application is running. Perform the below activities to understand the features of the application.



1. Click on the Login button at the top right corner and observe the URL.

2. Login with different credentials (other than admin, admin) and see the message displayed.

3. Login with credentials (admin, admin) and check how the redirection is happening by observing the URL.

4. Click on the two tabs (Tablets, Mobiles) which display tablet and mobile devices, respectively.

5. Click on any product name and see the product detail page getting displayed.

6. Click on Add to Cart button and add multiple products to the cart (selection count and the total price will be displayed on the second navigation bar).

7. Click on the cart link on the second navigation bar and observe the cart page which displays the selected products.

8. Click on the Checkout button and observe the page displayed. Click the Back button and observe the navigation.

9. Click on the sort dropdown and observe sort functionality based on the options mentioned.

10. Click on the filter dropdown and observe filtering functionality based on the options mentioned.

11. In the search text box placed on the second navigation bar, type the manufacturer name like Samsung, Apple, etc., and observe the search functionality.

12. Click on the Logout button at the top right corner and observe the redirection happening.

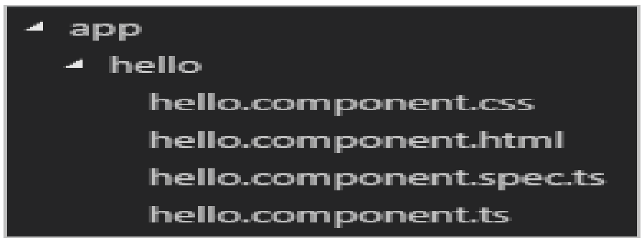
1. B)Course Name: Angular JS

Module Name: Components and Modules

Create a new component called hello and render Hello Angular on the page

Problem Statement: Creating a new component called hello and rendering Hello Angular on the page as shown below

1. In the same **MyApp** application created earlier, create a new component called hello using the following CLI command
2. `D:\MyApp> ng generate component hello`
3. This command will create a new folder with the name hello with the following files placed inside it



```
◢ app
   ◢ hello
       hello.component.css
       hello.component.html
       hello.component.spec.ts
       hello.component.ts
```

Open **hello.component.ts** file and create a property called courseName of type string and initialize it to "Angular" as shown below in Line number 9

```
1. import{Component,OnInit}from'@angular/core';
2.
3. @Component({
4. selector:'app-hello',
5. templateUrl:'./hello.component.html',
6. styleUrls:['./hello.component.css']
7. })
8. exportclassHelloComponentimplementsOnInit{
9. courseName:string="Angular";
10.
11.     constructor(){}
12.
13.     ngOnInit()
14.
```

Open **hello.component.html** and display the courseName as shown below in Line 2

```
1. <p>
2. Hello {{ courseName }}
3. </p>
```

Open **hello.component.css** and add the following styles for the paragraph element

Open **app.module.ts** file and add HelloComponent to bootstrap property as shown below in Line 11 to load it for execution

```
import{NgModule}from'@angular/core';
import{BrowserModule}from'@angular/platform-browser';

import{AppRoutingModule}from'./app-routing.module';
import{AppComponent}from'./app.component';
import{HelloComponent}from'./hello/hello.component';

@NgModule({
imports:[BrowserModule,AppRoutingModule],
     declarations:[AppComponent,HelloComponent],
     providers:[],
     bootstrap:[HelloComponent]
     })
     exportclassAppModule{}
```

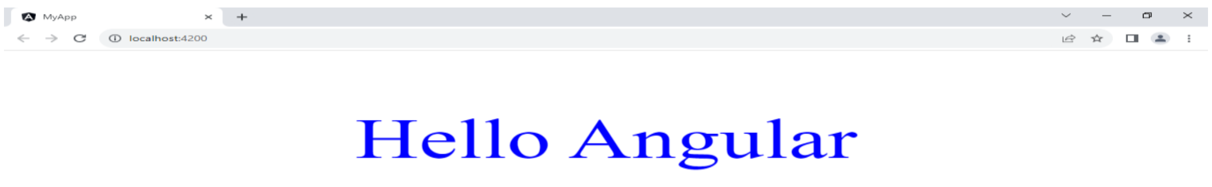
7. Open **index.html** and load the hello component by using its selector name i.e., app-hello as shown below in Line 11

```
1.  <!doctype html>
2.  <htmllang  "en">
3.  <head>
4.  <metacharset  "utf-8">
5.  <title>          </title>
6.  <basehref  "/">
7.  <metaname  "viewport"content  "width=device-width, initial-scale=1">
8.  <linkrel  "icon"type  "image/x-icon"href  "favicon.ico">
9.  </head>
10. <body>
11. <app-hello></app-hello>
12. </body>
13. </html>
```

8. Now run the application by giving the following command

```
D:\MyApp>ng serve --open
```

Output:-



Hello Angular

1. C) Course Name: Angular JS

Module Name: Elements of Template

Add an event to the hello component template and when it is clicked, it should change theCourseName.

Problem Statement: Adding an event to the hello component template and when it is clicked, it should change the courseName as shown below

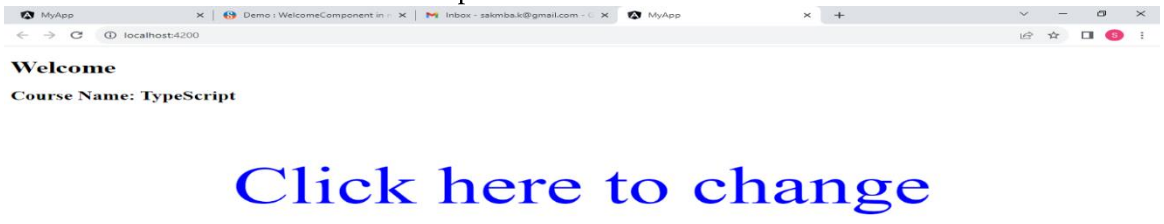
Open **hello.component.ts**, add a method called changeName() as shown below in Line 12-14. Also, use external template hello.component.html.

```
1.  import{Component,OnInit}from'@angular/core';
2.  @Component({
3.  selector:'app-hello',
4.  templateUrl:"./hello.component.html",
5.  styleUrls:['./hello.component.css']
6.  })
7.  exportclassHelloComponentimplementsOnInit{
8.  courseName="Angular";
9.  constructor(){}
10. ngOnInit(){
11. }
12. changeName(){
13. this.courseName="TypeScript";
14. }
15. }
```

Open **hello.component.html** and add a paragraph and bind it with changeName() method as shown in Line 3

```
1.  <h1>Welcome</h1>
2.  <h2>Course Name: {{ courseName }}</h2>
3.  <p (click)="changeName()">Click here to change</p>
```

2. Save the files and check the output in the browser



Welcome

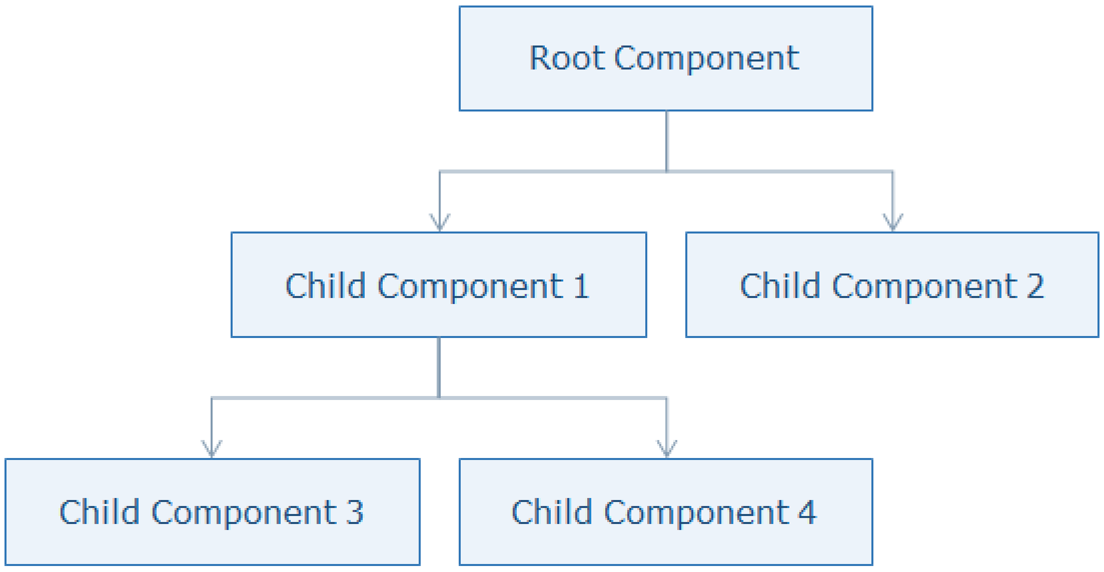
Course Name: TypeScript

Click here to change

1.d) Course Name: Angular JS
Module Name: Change Detection progressively building the PoolCarz application

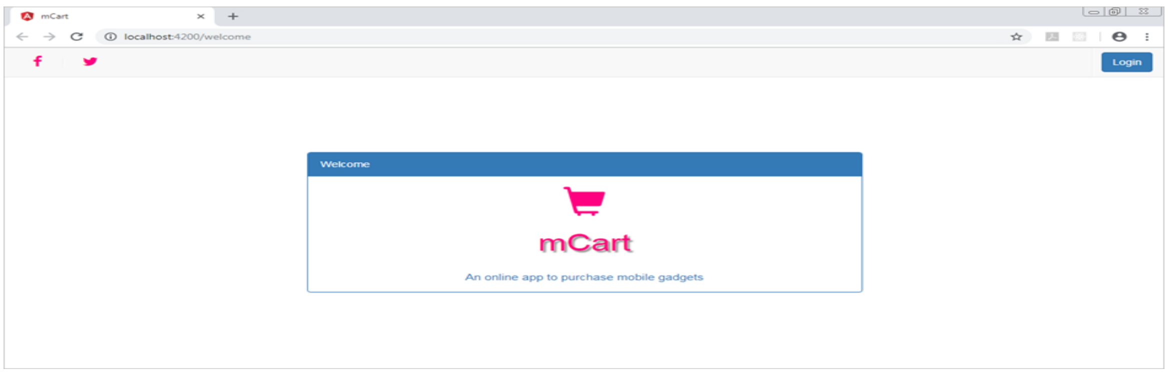
What does Angular do when a change is detected?

- Angular runs a change detector algorithm on each component from top to bottom in the component tree. This change detector algorithm is automatically generated at run time which will check and update the changes at appropriate places in the component tree.
- Angular is very fast though it goes through all components from top to bottom for every single event as it generates VM-friendly code. Due to this, Angular can perform hundreds of thousands of checks in a few milliseconds.



Demosteps:

- In the mCart application, there is a welcome screen displayed on application launch, as shown here.



- This welcome screen is created in the WelcomeComponent.

- You can find the files related to WelcomeComponent in the welcome folder present inside the app folder (src --> app --> welcome).

1. Code for WelcomeComponent is present in the file **welcome.component.ts.**

```
1.  import{Component}from'@angular/core';
2.
3.  @Component({
4.  templateUrl:'welcome.component.html',
5.  styleUrls:['welcome.component.css']
6.  })
7.  exportclassWelcomeComponent{
8.  publicpageTitle='Welcome';
9.
10. constructor(){
11.
12. }
13. }
```

Line 3-6: @Component marks the class as component and the component is bound with template and CSS file using templateUrl and styleUrls properties respectively.

Line 8: Creates a property called pageTitle and initialized it to “welcome”.

Line 11: This statement displays the login button at the top right corner of the page.

2. Code for WelcomeComponent template is present in the **welcome.component.html** file.

```
<divclass="container container-styles">
<divclass="panel panel-primary">
<divclass="panel-heading">{{pageTitle}}</div>
<divclass="panel-body">
<divclass="row">
<spanclass="img-responsive center-block logo-styles">
<spanclass="glyphiconglyphicon-shopping-cart"></span>
</span>
<divid="div1"class="shadow title-styles">mCart</div>

</div>
<br/>
<divclass="row">
<divclass="text-center text-styles">An online app to purchase mobile gadgets</div>
</div>
</div>
</div>
</div>
```

Line 4: pageTitle property is rendered using interpolation.

Line 7-9: Displays a shopping cart symbol. Used bootstrap CSS classes for this.

Line 10: Displays mCart title.

Code for the styling of WelcomeComponent is present in the **welcome.component.css** file.

```
1.  .shadow{
2.      text-shadow:3px3px2pxrgba(150,150,150,1);
3.  }
4.
5.  .logo-styles{
6.      width:50px;
7.      font-size:50px;
8.      color:#ff0080
9.  }
10.
11. .title-styles{
12.     text-align:center;
13.     color:#ff0080;
14.     font-size:40px
15. }
16.
17. .text-styles{
18.     color:#337ab7;
19.     font-size:15px
20. }
21.
22. .container-styles{
23.     position: relative;
24.     top:180px;
25.     width:50%
26. }
```

Line 1-3: shadow class applies a shadow effect to mCart text

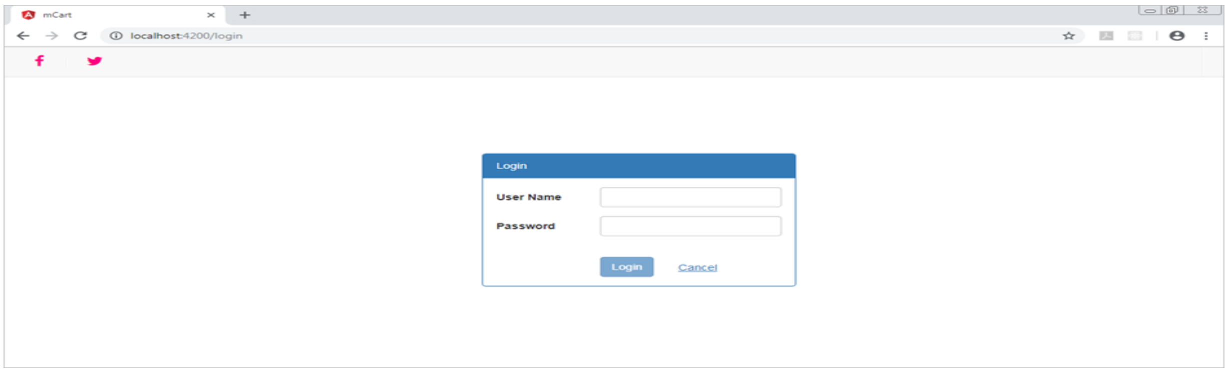
Line 5-9: logo-styles class applies width, font size and color properties to the shopping cart logo

Line 11-15: title-styles class applies the mentioned CSS properties to mCart text

Line 17-20: text-styles class applies the mentioned CSS properties to the description text rendered at the bottom of the welcome component

Line 22-26: container-styles class applies the mentioned CSS properties to the entire container

- In the mCart application, you have a login screen as shown here.



1. Code for the LoginComponent template is present in the **login.component.html** file.

```
<!-- Login form-->
<divclass="container container-styles">
<divclass="col-xs-7 col-xs-offset-3">
<divclass="panel panel-primary">
<divclass="panel-heading">Login</div>
<divclass="panel-body padding">
<formclass="form-horizontal" [formGroup]="loginForm">
<divclass="form-group">
<labelfor="name"class="col-xs-4 control-label"style="text-align:left">User Name</label>
<divclass="col-xs-8">
<inputtype="text"class="form-control" [ngClass]="{'valid':loginForm.controls['userName'].valid, 'invalid':loginForm.controls['userName'].invalid && !loginForm.controls['userName'].pristine}"formControlName="userName">
<div *ngIf="loginForm.controls['password'].errors&&loginForm.controls['userName'].dirty">
<div *ngIf="loginForm.controls['userName'].errors?.['required']"style="color:red">UserName is required
</div></div></div></div>
<divclass="form-group">
<labelfor="password"class="col-xs-4 control-label"style="text-align:left">Password</label>
<divclass="col-xs-8">
<inputtype="password"class="form-control" [ngClass]="{'valid':loginForm.controls['password'].valid, 'invalid':loginForm.controls['password'].invalid && !loginForm.controls['password'].pristine}"formControlName="password">
<div *ngIf="loginForm.controls['password'].errors&&loginForm.controls['password'].dirty">
<div *ngIf="loginForm.controls['password'].errors?.['required']"style="color:red">Password is required
</div></div></div></div>
```

```
22. <div *ngIf="!valid"class="error">Invalid Credentials...Please try again...</div>
23. <br/>
24.
25. <divclass="form-group">
26. <spanclass="col-xs-4"></span>
27. <divclass="col-xs-3">
28. <button (click)="onSubmit()"class="btnbtn-primary"
    [disabled]="!loginForm.valid">Login</button>
29. </div>
30. <spanclass="col-xs-5"style="top:8px">
31. <a [routerLink]="['/welcome']"style="color:#337ab7;text-decoration:
    underline;">Cancel</a>
32. </span></div></form></div></div></div></div>
```

Line 7-41: We have created a form with two labels and two text boxes for username and password

Note: In the code snippet (Line nos: 11-13 and 21-23), there are some additional code for data binding and validations which will be explained in subsequent demos.

Line 35: A login button is created and the click event is bound with the onSubmit() method. onSubmit() has the code to check the validity of the credentials.

Line 38: A hyperlink for canceling the operation and navigating back to the welcome component

Note: In the code snippet (Line no: 38), there is an additional code for routing which will be explained in subsequent demos.

LoginComponent has a model class coded in the **login.ts,** below is the code for it.

```
1. export class Login {
2. public userName: string='';
3. public password: string='';
4. }
```

Line 1-4: The Login class has two properties username and password to store the values entered by the user in the Login form.

Code for LoginComponent is coded in the file **login.component.ts.**

```
1. import{Component,ElementRef,OnInit,Renderer2,ViewChild}from'@angular/core';
2. import{FormBuilder,FormGroup,Validators}from'@angular/forms';
3. import{Router}from'@angular/router';
4. import{Login}from'./Login';
5. import{LoginService}from'./login.service';
6.
7. @Component({
8. templateUrl:'./login.component.html',
9. styleUrls:['./login.component.css'],
10. providers:[LoginService]
11. })
12. exportclassLoginComponentimplementsOnInit{
13.
14. login =newLogin();
15.
16.    ...
17.    onSubmit(){
```

Line 4: Importing Login class from login.ts

Line 7-11: Component decorator marks the class as a component and templateUrl to bind HTML page to Login component

Line 14: An instance of Login class is created

Finally, bind both the components of Welcome and Login in the root module.

This is done in the module file of the application i.e. **app.module.ts** file, it contains the below-given code.

```
1.  import { NgModule } from '@angular/core';
2.  import { BrowserModule } from '@angular/platform-browser';
3.  import { ReactiveFormsModule } from '@angular/forms';
4.  import { HttpClientModule } from '@angular/common/http';
5.
6.  import { AppComponent } from './app.component';
7.  import { AppRoutingModule } from './app-routing.module';
8.  import { WelcomeComponent } from './welcome/welcome.component';
9.  import { LoginComponent } from './login/login.component';
10.
11. @NgModule({
12. imports: [BrowserModule, HttpClientModule, ReactiveFormsModule,
    AppRoutingModule],
13. declarations: [AppComponent, WelcomeComponent, LoginComponent],
14. providers: [],
15. bootstrap: [AppComponent]
16. })
17. export class AppModule{ }
```

Line 7-8: Import WelcomeComponent and LoginComponent classes

Line 13: Include them in the declarations property of NgModule decorator to make them available in the entire module

2. A) Module Name: Structural Directives –ngIf (MCART)

Create a login form with username and password fields. If the user enters the correct credentials, it should render a "Welcome <<username>>" message otherwise it should render "Invalid Login!!! Please try again..." message

1. Open **app.component.ts and** write the following code:

```
import{Component}from'@angular/core';

@Component({
selector:'app-root',
templateUrl:'./app.component.html',
styleUrls:['./app.component.css'],
})
exportclassAppComponent{
isAuthenticated!:boolean;
submitted =false;
userName!: string;

onSubmit(name: string, password: string){
this.submitted=true;
this.userName= name;
if(name ==='admin'&& password ==='admin'){
this.isAuthenticated=true;
}else{
this.isAuthenticated=false;
}
}
}
```

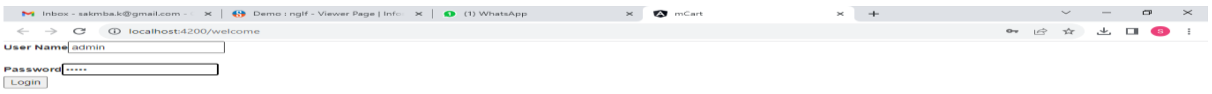
2. Write the below-given code in **app.component.html:**

```
div *ngIf="!submitted">
<form>
<label>User Name</label>
<inputtype="text" #username/><br/><br/>
<labelfor="password">Password</label>
<inputtype="password"name="password" #password/><br/>
</form>
<button (click)="onSubmit(username.value, password.value)">Login</button>
</div>

<div *ngIf="submitted">
<div *ngIf="isAuthenticated; else failureMsg">
<h4>Welcome {{ userName }}</h4>
</div>
<ng-template #failureMsg>
<h4>Invalid Login !!! Please try again...</h4>
</ng-template>
<buttontype="button" (click)="submitted = false">Back</button>
</div>
```

3. Add AppComponent to the bootstrap property in the root module file i.e., **app.module.ts**

```
import{BrowserModule}from'@angular/platform-browser';
import{NgModule}from'@angular/core';

import{AppComponent}from'./app.component';

@NgModule({
declarations:[
AppComponent
],
imports:[
BrowserModule
],
providers:[],
bootstrap:[AppComponent]
})
exportclassAppModule{}
```

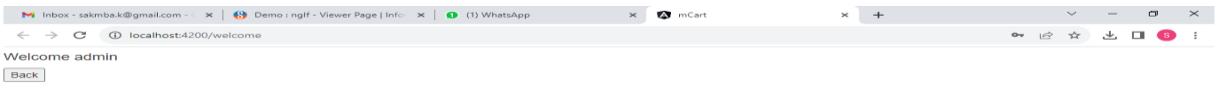
OUTPUT: -



User Name admin

Password •••••

Login



Welcome admin

Back

2.b **Course Name:** Angular JS

Module Name: ngFor

Problem Statement : Create a courses array and rendering it in the template using ngFor directive in a list format.

Solution:

ngFor directive is used to iterate over-collection of data i.e., arrays

syntax:

ngFor = "expression"
Example:

app.component.ts

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
templateUrl: './app.component.html',
styleUrls: ['./app.component.css']
})
export class AppComponent {
  courses: any[] = [
{ id: 1, name: 'TypeScript' },
{ id: 2, name: 'Angular' },
{ id: 3, name: 'Node JS' },
{ id: 1, name: 'TypeScript' }
  ];
}
```

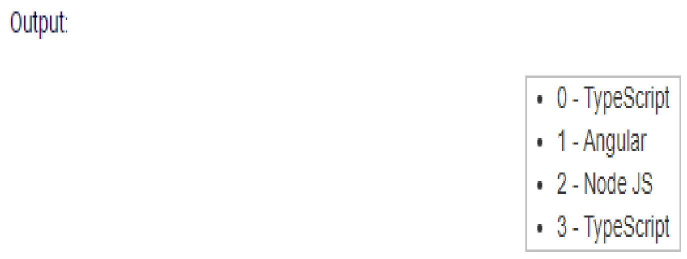
- Line 3-8: Creating an array of objects

app.component.html

```
<ul>
<li *ngFor="let course of courses;  leti = index">
       {{i}} - {{ course.name }}
</li>
</ul>
```

- Line 2: ngFor iterates over courses array and displays the value of name property of each course. It also stores the index of each item in a variable called i
- Line 3: {{ i }} displays the index of each course and course.name displays the name property value of each course

Output:


Output:

```
• 0 - TypeScript
• 1 - Angular
• 2 - Node JS
• 3 - TypeScript
```

2.c Course Name: Angular JS

Module Name: ngSwitch

Problem Statement : Display the correct option based on the value passed to ngSwitch directive.

Solution:

Write the below-given code in **app.component.ts**

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
templateUrl: './app.component.html',
styleUrls: ['./app.component.css']
})
export class AppComponent {
  choice = 0;
nextChoice() {
this.choice++;
  }
}
```

- Line 3: Creates a choice property and initializes it to zero
- Line 5-7: nextChoice() increments the choice when invoked

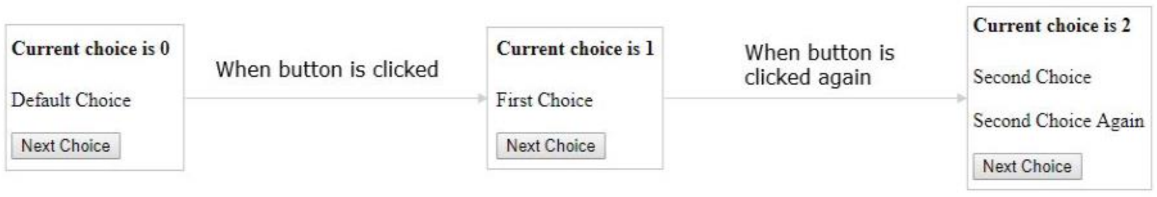
Write the below-given code in **app.component.html**

```
<h4>
  Current choice is {{ choice }}
</h4>
<div [ngSwitch]="choice">
<p *ngSwitchCase="1">First Choice</p>
<p *ngSwitchCase="2">Second Choice</p>
<p *ngSwitchCase="3">Third Choice</p>
```

```
<p *ngSwitchCase="2">Second Choice Again</p>
<p *ngSwitchDefault>Default Choice</p>
</div>
<div>
<button (click)="nextChoice()">
            Next Choice
</button>
</div>
```

- Line 2: ngSwitch takes the value and based upon the value inside the choice property, it executes *ngSwitchCase. Paragraph elements will be added/removed from the DOM based on the value passed to the switch case.

Output:



2.d **Course Name:** Angular JS

Module Name: Custom Structural Directive

Problem Statement : Create a custom structural directive called 'repeat' which should repeat the element given a number of times.

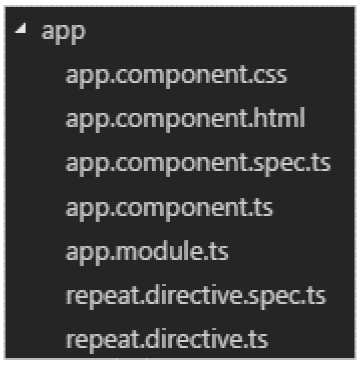
To create a custom structural directive, create a class annotated with @Directive

```
   @Directive({
})
class MyDirective{}
```

Generate a directive called 'repeat' using the following command

```
  D:\MyApp>ng generate directive repeat
```

This will create two files under the src\app folder with names repeat.directive.ts and repeat.directive.spec.ts (this is for testing). Now the app folder structure will look as shown below:



```
◢ app
    app.component.css
    app.component.html
    app.component.spec.ts
    app.component.ts
    app.module.ts
    repeat.directive.spec.ts
    repeat.directive.ts
```

It also adds repeat directive to the root module i.e., app.module.ts to make it available to the entire module as shown below in Line 7
Write the below-given code in app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';
import { RepeatDirective } from './repeat.directive';
@NgModule({
  declarations: [
AppComponent,
RepeatDirective
  ],
  imports: [
BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
```

```
export class AppModule{ }
```

Open **repeat.directive.ts** file and add the following code

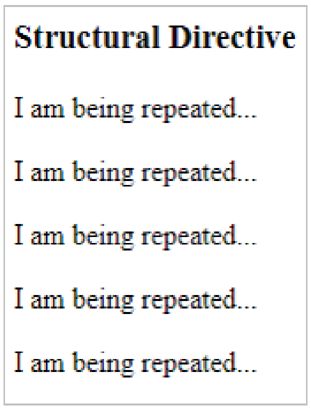
```
import { Directive, TemplateRef, ViewContainerRef, Input } from '@angular/core';
@Directive({
  selector: '[appRepeat]'
})
export class RepeatDirective {
constructor(private templateRef: TemplateRef<any>, private viewContainer: ViewContainerRef) { }
  @Input() set appRepeat(count: number) {
    for (let i = 0; i< count; i++) {
this.viewContainer.createEmbeddedView(this.templateRef);
    }
  }
}
```

Line 3: Annotate the class with @Directive which represents the class as a directive and specify the selector name inside brackets
Line 8: Create a constructor and inject two classes called TemplateRef which acquires <ng-template> content and another class called ViewcontainerRef which access the HTML container to add or remove elements from it
Line 10: Create a setter method for an appRepeat directive by attaching @Input() decorator which specifies that this directive will receive value from the component. This method takes the number passed to the appRepeat directive as an argument.
Line 12: As we need to render the elements based on the number passed to the appRepeat directive, run a for loop in which pass the template reference to a createEmbeddedView method which renders the elements into the DOM. This structural directive creates an embedded view from the Angular generated <ng-template> and inserts that view in a view container.

Write the below-given code in app.component.html

```
<h3>Structural Directive</h3>
<p *appRepeat="5">I am being repeated...</p>
```

Output:



```
Structural Directive

I am being repeated...

I am being repeated...

I am being repeated...

I am being repeated...

I am being repeated...
```

3.a Course Name: Angular JS

Module Name: Attribute Directives – ngStyle

Problem Statement : Apply multiple CSS properties to a paragraph in a component using ngStyle.

Solution:

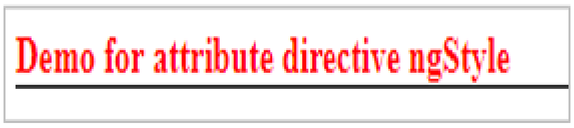
Write the below-given code in app.component.ts

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
templateUrl: './app.component.html',
styleUrls: ['./app.component.css']
})
export class AppComponent {
colorName = 'red';
fontWeight = 'bold';
borderStyle = '1px solid black';
}
```

Write the below-given code in app.component.html

```
<p [ngStyle]="{
color:colorName,
            'font-weight':fontWeight,
borderBottom: borderStyle
          }">
  Demo for attribute directive ngStyle
</p>
```

Output:



Demo for attribute directive ngStyle

3.b Course Name: Angular JS

Module Name: ngClass

Problem Statement : Apply multiple CSS classes to the text using ngClass directive.

Solution:

Write the below-given code in app.component.ts

```
 import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
templateUrl: './app.component.html',
styleUrls: ['./app.component.css']
})
export class AppComponent {
isBordered = true;
}
```

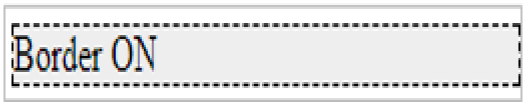
Write the below-given code in app.component.html

```
 <div [ngClass]="{bordered: isBordered}">
  Border {{ isBordered ? "ON" : "OFF" }}
</div>
```

In app.component.css, add the following CSS class

```
.bordered {
        border: 1px dashed black;
        background-color: #eee;
}
```

Output:



```
Border ON
```

3.c Course Name: Angular JS

Module Name: Custom Attribute Directive

Problem Statement : Create an attribute directive called 'showMessage' which should display the givenmessage in a paragraph when a user clicks on it and should change the text color to red

Solution:

Generate a directive called 'message' using the following command

D:\MyApp>ng generate directive message

Above command will add MessageDirective class to the declarations property in the app.module.ts **file**

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';
import { MessageDirective } from './message.directive';
@NgModule({
  declarations: [
AppComponent,
MessageDirective
  ],
  imports: [
BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule{ }
```

Open the message.directive.ts **file and add the following code**

```
import { Directive, ElementRef, Renderer2, HostListener, Input } from '@angular/core';
@Directive({
  selector: '[appMessage]',
})
export class MessageDirective {
  @Input('appMessage') message!: string;
constructor(private el: ElementRef, private renderer: Renderer2) {
renderer.setStyle(el.nativeElement, 'cursor', 'pointer');
  }
  @HostListener('click') onClick() {
this.el.nativeElement.innerHTML = this.message;
this.renderer.setStyle(this.el.nativeElement, 'color', 'red');
  }
}
```

Write the below-given code in app.component.html

```
<h3>Attribute Directive</h3>
<p [appMessage]="myMessage">Click Here</p>
```

Add the following CSS styles to the app.component.css **file**

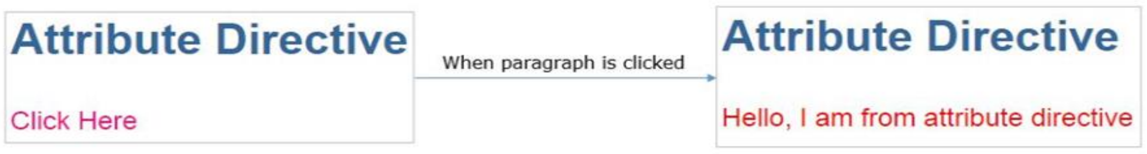
```
h3 {
  color: #369;
  font-family: Arial, Helvetica, sans-serif;
  font-size: 250%;
}
p {
  color: #ff0080;
  font-family: Arial, Helvetica, sans-serif;
  font-size: 150%;
}
```

Add the following code in app.component.ts

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
templateUrl: './app.component.html',
styleUrls: ['./app.component.css']
})
```

```
export class AppComponent {
myMessage = 'Hello, I am from attribute directive';
}
```

Output:


Attribute Directive
Click Here
When paragraph is clicked
Attribute Directive
Hello, I am from attribute directive

4.a Course Name: Angular JS
Module Name: Property Binding

Problem Statement : Binding image with class property using property binding.

Solution:
Write the following code in app.component.ts as shown below

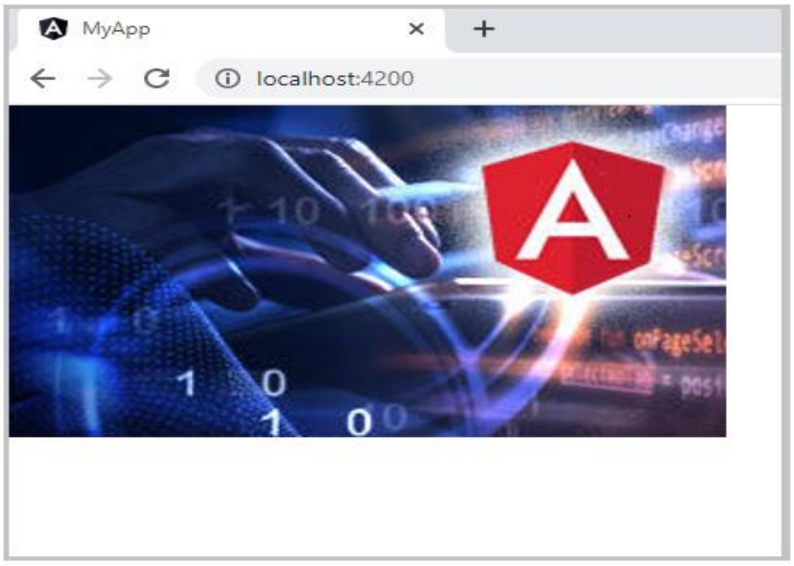
```
import { Component } from '@angular/core';

@Component({

  selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css']

})

export class AppComponent {

imgUrl = 'assets/imgs/logo.png';

}
```

Create a folder named "imgs" inside src/assets and place a logo.png file inside it.

Write the following code in app.component.html as shown below

```
<img [src]='imgUrl'>
```

Output:



4.b Course Name: Angular JS

Module Name: Attribute Binding

Problem Statement : Binding colspan attribute of a table element to the class property..

Solution:

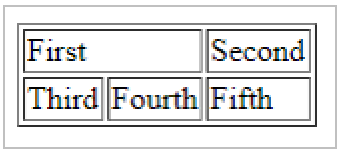
Write the below-given code in **app.component.ts**

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
templateUrl: './app.component.html',
styleUrls: ['./app.component.css']
})
export class AppComponent {
colspanValue = '2';
}
```

Write the below-given code in **app.component.html**

```
<table border=1>
<tr>
<td [attr.colspan]="colspanValue"> First </td>
<td>Second</td>
</tr><tr>
<td>Third</td>
<td>Fourth</td>
<td>Fifth</td>
</tr></table>
```

output:



First		Second
Third	Fourth	Fifth

4.c Course Name: Angular JS

Module Name: Style and Event Binding

Problem Statement : Binding an element using inline style and user actions like entering text in input fields.

Solution:

Write the below-given code in **app.component.ts**

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
templateUrl: './app.component.html',
styleUrls: ['./app.component.css']
})
export class AppComponent {
  name = 'Angular';
}
```

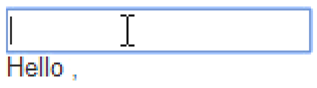
Write the below-given code in **app.component.html**

```
<input type="text" [(ngModel)]="name"><br/>
<div>Hello , {{ name }}</div>
```

Write the below-given code in **app.module.ts**

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
import { AppComponent } from './app.component';
@NgModule({
  declarations: [
AppComponent
  ],
  imports: [
BrowserModule,
FormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule{ }
```

output:



```
Hello ,
```

5.a Course Name: Angular JS

Module Name: Built in Pipes

Problem Statement : Display the product code in lowercase and product name in uppercase using built-in pipes

Solution:

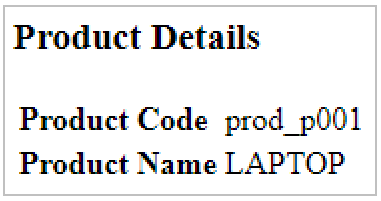
Write the below-given code in **app.component.ts**

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
templateUrl: './app.component.html',
styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'product details';
productCode = 'PROD_P001';
productName = 'Laptop';
}
```

Write the below-given code in **app.component.html**

```
<h3>{{ title | titlecase}} </h3>
<table style="text-align:left">
<tr>
<th> Product Code </th>
<td>{{ productCode | lowercase }} </td>
</tr>
<tr>
<th> Product Name </th>
<td>{{ productName | uppercase }} </td>
</tr>
</table>
```

Output:



```
Product Details

Product Code  prod_p001
Product Name LAPTOP
```

5.b Course Name: Angular JS

Module Name: Passing Parameters to Pipes

Problem Statement : Apply built-in pipes with parameters to display product details.

Solution:

Write the below-given code in **app.component.ts**

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
templateUrl: './app.component.html',
styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'product details';
productCode = 'PROD_P001';
productName = 'Apple MPTT2 MacBook Pro';
productPrice = 217021;
purchaseDate = '1/17/2018';
productTax = '0.1';
productRating = 4.92;
}
```

Write the below-given code in **app.component.html**

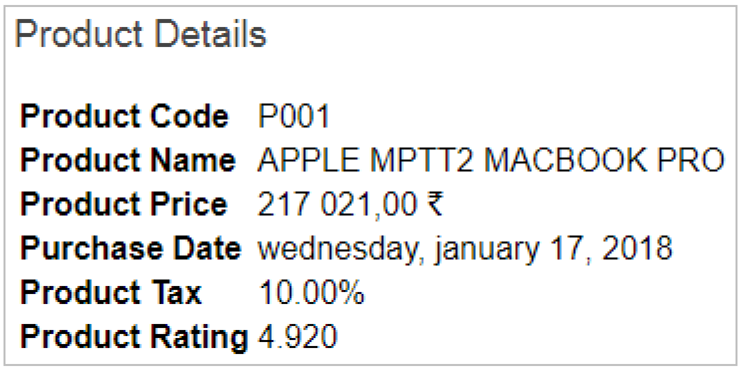
```
<h3>{{ title | titlecase}} </h3>
<table style="text-align:left">
<tr>
<th> Product Code </th>
<td>{{ productCode | slice:5:9 }} </td>
</tr>
<tr>
<th> Product Name </th>
<td>{{ productName | uppercase }} </td>
</tr>
<tr>
<th> Product Price </th>
<td>{{ productPrice | currency: 'INR':'symbol':'':'fr' }} </td>
</tr>
<tr>
<th> Purchase Date </th>
<td>{{ purchaseDate | date:'fullDate' | lowercase}} </td>
</tr>
<tr>
<th> Product Tax </th>
<td>{{ productTax | percent : '.2' }} </td>
</tr>
<tr>
<th> Product Rating </th>
```

```
<td>{{ productRating | number:'1.3-5'}} </td>
</tr>
</table>
```

Write the below-given code in **app.module.ts**

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';
import { registerLocaleData } from '@angular/common';
import localeFrench from '@angular/common/locales/fr';
registerLocaleData(localeFrench);
@NgModule({
  declarations: [
AppComponent
  ],
  imports: [
BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule{ }
```

Output:



```
Product Details

Product Code    P001
Product Name    APPLE MPTT2 MACBOOK PRO
Product Price   217 021,00 ₹
Purchase Date   wednesday, january 17, 2018
Product Tax     10.00%
Product Rating  4.920
```

We have applied currency pipe to product price with locale setting as 'fr' i.e., French. According to the French locale, the currency symbol will be displayed at the end of the price as shown in the above output.

5.c Course Name: Angular JS

Module Name: Nested Components Basics

Problem Statement : Load Courses list Component in the root component when a user clicks on the View courses list button

Solution:

1. Create a component called coursesList using the following CLI command

```
D:\MyApp>ng generate component coursesList
```

The above command will create a folder with name courses-list with the following files

- courses-list.component.ts
- courses-list.component.html
- courses-list.component.css
- courses-list.component.spec.ts

2. CoursesListComponent class will be added in the **app.module.ts** file

```
import { BrowserModule } from '@angular/platform-browser';

import { NgModule } from '@angular/core';

import { AppComponent } from './app.component';

import { CoursesListComponent } from './courses-list/courses-list.component';

@NgModule({

  declarations: [

AppComponent,

CoursesListComponent

],  imports: [

BrowserModule

],  providers: [],

  bootstrap: [AppComponent]

})export class AppModule { }
```

3. Write the below-given code in **courses-list.component.ts**

```
import { Component, OnInit } from '@angular/core';

@Component({

  selector: 'app-courses-list',

templateUrl: './courses-list.component.html',
styleUrls: ['./courses-list.component.css']
})
export class CoursesListComponent {
  courses = [
{ courseId: 1, courseName: "Node JS" },
{ courseId: 2, courseName: "Typescript" },
{ courseId: 3, courseName: "Angular" },
{ courseId: 4, courseName: "React JS" }
  ];
}
```

4. Write the below-given code in **courses-list.component.html**

```
<table border="1">
<thead>
<tr>
<th>Course ID</th>
<th>Course Name</th>
</tr>
</thead>
<tbody>
<tr *ngFor="let course of courses">
<td>{{ course.courseId }}</td>
<td>{{ course.courseName }}</td>
</tr>
</tbody>
</table>
```

5. Add the following code in **courses-list.component.css**

```
tr{
text-align:center;
}
```

6. Write the below-given code in **app.component.html**

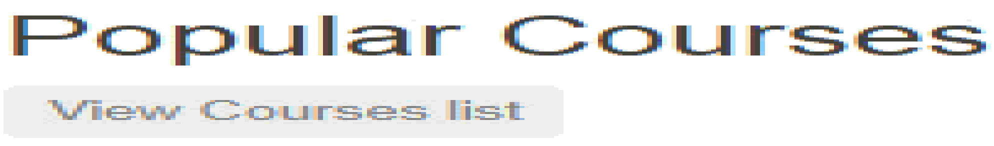
```
<h2>Popular Courses</h2>

<button (click)="show = true">View Courses list</button><br /><br />

<div *ngIf="show">

<app-courses-list></app-courses-list>

</div>
```

7. Write the below-given code in **app.component.ts**

```
import { Component } from '@angular/core';
@Component({

        selector: 'app-root',
templateUrl: './app.component.html',
styleUrls: ['./app.component.css']
})
export class AppComponent {
show!:boolean;
}
```

8. Save the files and check the output in the browser

OUTPUT:-



6.A)Course Name: Angular JS
Module Name: Passing data from Container Component to Child Component

PROBLEM STATEMENT: Create an AppComponent that displays a dropdown with a list of courses as values init. Create another component called the CoursesList component and load it in AppComponent which should display the course details. When the user selects a course from the list.

1. Open the **courses-list.component.ts** file created in the example of nested components and add the following code

```
import { Component, Input } from '@angular/core';

@Component({

  selector: 'app-courses-list',

templateUrl: './courses-list.component.html',

styleUrls: ['./courses-list.component.css'],

})

export class CoursesListComponent {

  courses = [

{ courseId: 1, courseName: 'Node JS' },

{ courseId: 2, courseName: 'Typescript' },

{ courseId: 3, courseName: 'Angular' },

{ courseId: 4, courseName: 'React JS' },

  ];

course!: any[];

  @Input() set cName(name: string) {

this.course = [];

    for (var i = 0; i<this.courses.length; i++) {

      if (this.courses[i].courseName === name) {

this.course.push(this.courses[i]);

      }

    }

  }

}
```

2. Open **courses-list.component.html** and add the following code

```
<table border="1" *ngIf="course.length> 0">
<thead>
<tr>
<th>Course ID</th>
<th>Course Name</th>
</tr>
</thead>
<tbody>
<tr *ngFor="let c of course">
<td>{{ c.courseId }}</td>
<td>{{ c.courseName }}</td>
</tr>
</tbody>
</table>
```

3. Add the following in **app.component.html**

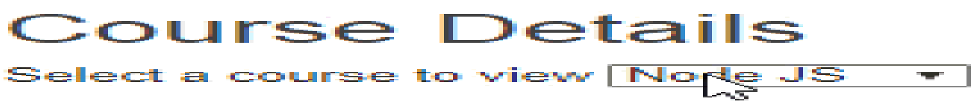
```
<h2>Course Details</h2>
Select a course to view
<select #course (change)="name = course.value">
<option value="Node JS">Node JS</option>
<option value="Typescript">Typescript</option>
<option value="Angular">Angular</option>
<option value="React JS">React JS</option></select><br /><br />
<app-courses-list [cName]="name"></app-courses-list>
```

4. Add the following in **app.component.ts**

```
import { Component } from '@angular/core';
@Component({
  selector: 'app-root',
styleUrls: ['./app.component.css'],
templateUrl: './app.component.html'
})
export class AppComponent {
```

```
name!: string;

}
```

OUTPUT:-



Course Details

Select a course to view Node JS

6.B)Course Name: Angular JS

Module Name: Passing data from Child Component to ContainerComponent

Create an AppComponent that loads another component called the CoursesListcomponent. Create another component called CoursesListComponent which shoulddisplay the courses list in a table along with a register .button in each row. When auser clicks on

1. Open the **courses-list.component.ts** file created in the previous example and add the following code

```
import { Component, OnInit, Input, Output, EventEmitter } from '@angular/core';
@Component({
  selector: 'app-courses-list',
templateUrl: './courses-list.component.html',
styleUrls: ['./courses-list.component.css']
})
export class CoursesListComponent {
  @Output() registerEvent = new EventEmitter<string>();
  courses = [
{ courseId: 1, courseName: 'Node JS' },
{ courseId: 2, courseName: 'Typescript' },
{ courseId: 3, courseName: 'Angular' },
{ courseId: 4, courseName: 'React JS' }
  ];
register(courseName: string) {

this.registerEvent.emit(courseName);  }}
```

2. Open **courses-list.component.html** and add the following code

```
<table border="1">
<thead>
<tr>
<th>Course ID</th>
<th>Course Name</th>
<th></th>
</tr>
</thead>
<tbody>
<tr *ngFor="let course of courses">
<td>{{ course.courseId }}</td>
<td>{{ course.courseName }}</td>
<td><button (click)="register(course.courseName)">Register</button></td>
</tr>
</tbody>
</table>
```

3. Add the following in **app.component.html**

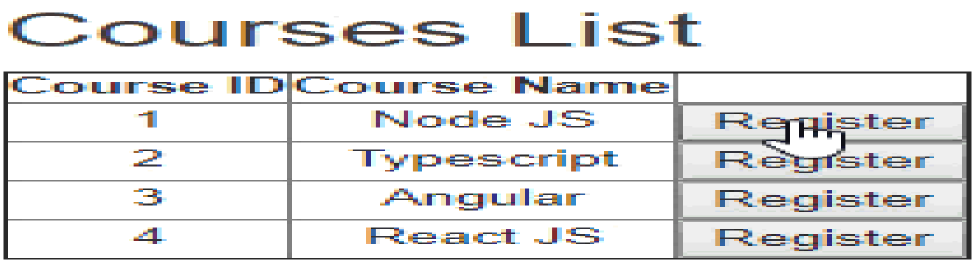
```
<h2>Courses List</h2>
```

```
<app-courses-list (registerEvent)="courseReg($event)"></app-courses-list>
<br /><br />

<div *ngIf="message">{{ message }}</div>
```

4. Add the following code in **app.component.ts**

```
import { Component } from '@angular/core';

@Component({

  selector: 'app-root',

templateUrl: './app.component.html',

styleUrls: ['./app.component.css']

})

export class AppComponent {

message!: string;

courseReg(courseName: string) {

this.message = `Your registration for ${courseName} is successful`;

  }

}
```

OUTPUT:-



Courses List

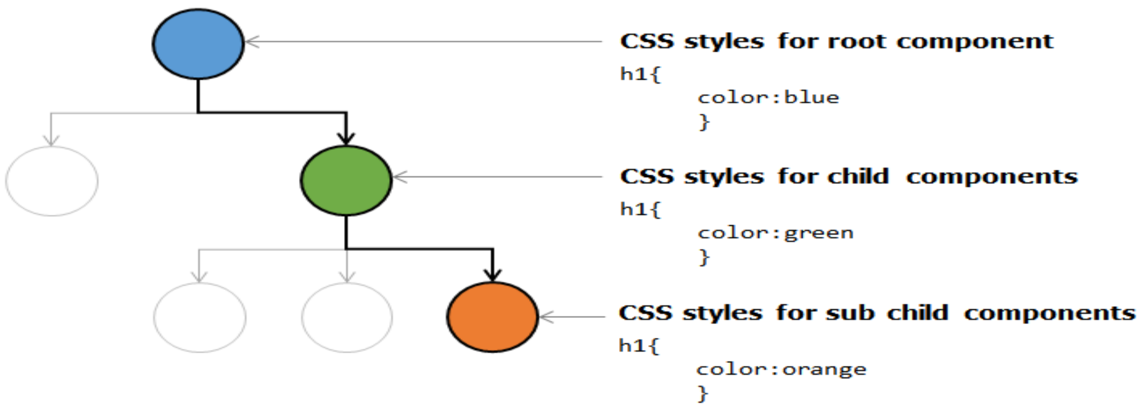
Course ID	Course Name	
1	Node JS	Register
2	Typescript	Register
3	Angular	Register
4	React JS	Register

6 C)Course Name: Angular JS
Module Name: Shadow DOM
Apply ShadowDOM and None encapsulation modes to components.

Shadow DOM is a web components standard by W3C. It **enables encapsulation for DOM tree and styles**. Shadow DOM hides DOM logic behind other elements and confines styles only for that component.

For example, in an Angular application, n number of components will be created and each component will have its own set of data and CSS styles. When these are integrated, there is a chance that the data and styles may be applied to the entire application. Shadow DOM encapsulates data and styles for each component to not flow through the entire application.

In the below example shown, each component is having its own styles defined and they are confined to themselves:


CSS styles for root component
h1{
color:blue
}
CSS styles for child components
h1{
color:green
}
CSS styles for sub child components
h1{
color:orange
}

1. Create a component called **First** using the following CLI command

```
D:\MyApp>ng generate component first
```

2. Write the below-given code in **first.component.css**

```
.cmp {
  padding: 6px;
  margin: 6px;
  border: blue 2px solid; }
```

3. Write the below-given code in **first.component.html**

```
<div class="cmp">First Component</div>
```

4. Create a component called **Second** using the following CLI command

```
D:\MyApp>ng generate component second
```

5. Write the below-given code in **second.component.css**

```
.cmp {
  border: green 2px solid;
  padding: 6px;
  margin: 6px;
}
```

5. Write the below-given code in **second.component.html**

```
<div class="cmp">Second Component</div>
```

6. Write the below-given code in **second.component.ts**

```
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
 selector: 'app-second',
templateUrl: './second.component.html',
styleUrls: ['./second.component.css'],
 encapsulation: ViewEncapsulation.ShadowDom
})
export class SecondComponent {
}
```

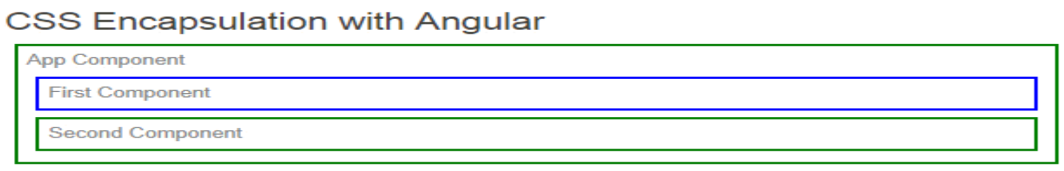
8. Write the below-given code in **app.component.css**

```
.cmp {
 padding: 8px;
 margin: 6px;
 border: 2px solid red;}
```

9. Write the below-given code in **app.component.html**

```
<h3>CSS Encapsulation with Angular</h3>
<div class="cmp">
  App Component
<app-first></app-first>
<app-second></app-second>
</div>
```

10 . Save the files and check the output in the browser


CSS Encapsulation with Angular

App Component

First Component

Second Component

1. Set ViewEncapsulation to none mode in **app.component.ts** file

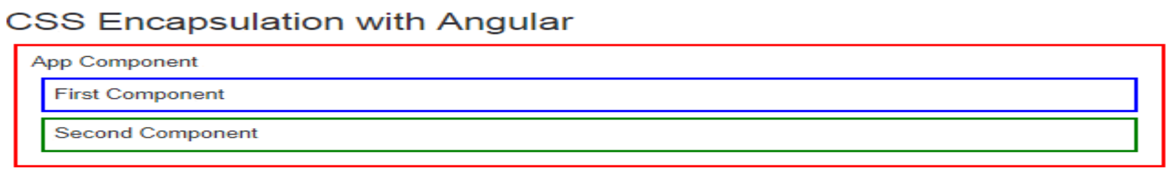
```
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
  selector: 'app-root',
styleUrls: ['./app.component.css'],
templateUrl: './app.component.html',
  encapsulation: ViewEncapsulation.None
})
export class AppComponent {
}
```

2. Set ViewEncapsulation to none mode in second.component.ts file

```
import { Component, ViewEncapsulation } from '@angular/core';
@Component({
  selector: 'app-second',
templateUrl: './second.component.html',
styleUrls: ['./second.component.css'],
  encapsulation: ViewEncapsulation.None
})export class SecondComponent {
```

}3. Save the files and check the output in the browser

OUTPUT:-


CSS Encapsulation with Angular

App Component

First Component

Second Component

6.D) Course Name: Angular JS
Module Name: Component Life Cycle
Override component life-cycle hooks and logging the corresponding messages to understand the flow.

<u>PROGRAM:-</u>

1. Write the below-given code in **app.component.ts**

```
import {
   Component, OnInit, DoCheck, AfterContentInit, AfterContentChecked,
AfterViewInit, AfterViewChecked,
OnDestroy
} from '@angular/core';
@Component({
   selector: 'app-root',
styleUrls: ['./app.component.css'],
templateUrl: './app.component.html'
})
export class AppComponent implements OnInit, DoCheck,
AfterContentInit, AfterContentChecked,
AfterViewInit, AfterViewChecked,
OnDestroy {
   data = 'Angular';
ngOnInit() {
      console.log('Init');
   }
ngDoCheck(): void {
console.log('Change detected');
   }
ngAfterContentInit(): void {
console.log('After content init');
   }
ngAfterContentChecked(): void {
```

```
console.log('After content checked');
   }
ngAfterViewInit(): void {
console.log('After view init');
   }
ngAfterViewChecked(): void {
console.log('After view checked');
   }
ngOnDestroy(): void {
       console.log('Destroy');
   }
}
```

2. Write the below-given code in **app.component.html**

```
<div>
<h1>I'm a container component</h1>
<input type="text" [(ngModel)]="data" />
<app-child [title]="data"></app-child>
</div>
```

3. Write the below-given code in **child.component.ts**

```
import { Component, OnChanges, Input } from '@angular/core';
@Component({
  selector: 'app-child',
templateUrl: './child.component.html',
styleUrls: ['./child.component.css']
})
export class ChildComponent implements OnChanges {
  @Input() title!: string;
ngOnChanges(changes: any): void {

console.log('changes in child:' + JSON.stringify(changes));  }}
```

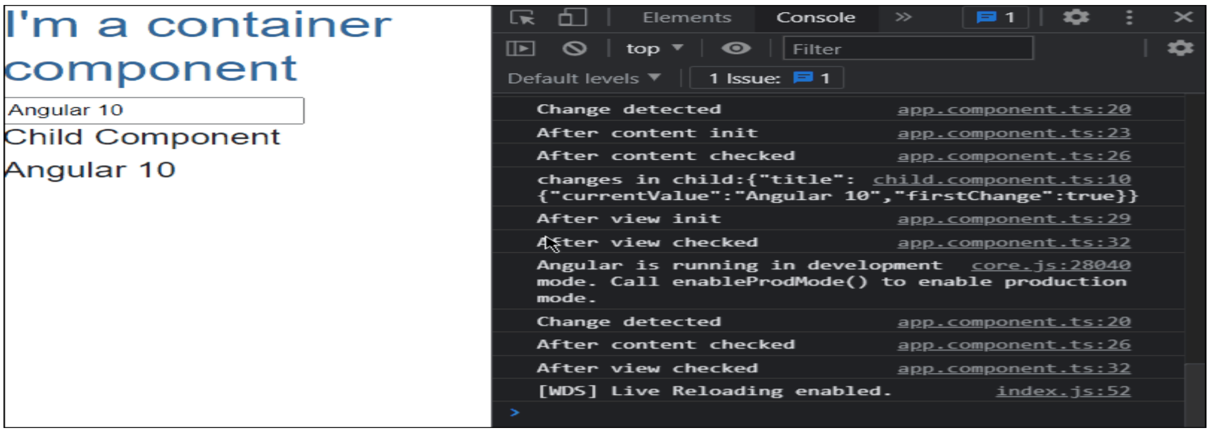
4. Write the below-given code in **child.component.html**

```
<h2>Child Component</h2>

<h2>{{title}}</h2>
```

5. Ensure FormsModule is present in the imports section of the AppModule.

6. Save the files and check the output in the browser

OUTPUT:-



```
Change detected                                    app.component.ts:20
After content init                                 app.component.ts:23
After content checked                              app.component.ts:26
changes in child:{"title":                       child.component.ts:10
{"currentValue":"Angular 10","firstChange":true}}
After view init                                    app.component.ts:29
After view checked                                 app.component.ts:32
Angular is running in development                   core.js:28040
mode. Call enableProdMode() to enable production
mode.
Change detected                                    app.component.ts:20
After content checked                              app.component.ts:26
After view checked                                 app.component.ts:32
[WDS] Live Reloading enabled.                           index.js:52
>
```

7 A)Course Name: Angular JS
Module Name: Template Driven Forms
Create a course registration form as a template-driven form.

Add the following code in the **registration-form.component.ts** file

```
import { Component, OnInit } from '@angular/core';

import { FormBuilder, FormGroup, Validators } from '@angular/forms';

@Component({

  selector: 'app-registration-form',

templateUrl: './registration-form.component.html',

styleUrls: ['./registration-form.component.css']

})

export class RegistrationFormComponent implements OnInit {

registerForm!:FormGroup;

submitted!:boolean;

constructor(private formBuilder: FormBuilder) { }

ngOnInit() {

this.registerForm = this.formBuilder.group({

firstName: ['', Validators.required],

lastName: ['', Validators.required],

      address: this.formBuilder.group({

        street: [],

        zip: [],

        city: []

      })    });  }}
```

Line 9: Create a property registerForm of type FormGroup

Line 11: Inject a FormBuilder instance using constructor

Line 13: formBuilder.group() method creates a FormGroup. It takes an object whose keys are FormControl names and values are their definitions

Line 14-20: Create form controls such as firstName, lastName, and address as a subgroup with fields street, zip, and city. These fields are form controls.

For each form control:

- you can mention the default value as the first argument and
- the list of validators as the second argument.
- Validations can be added to the form controls using the built-in validators supplied by the Validators class.
- For example: Configure built-in required validator for each control using [' ', Validators.required] syntax.
- If multiple validators are to be applied, then use the syntax [' ', [Validators.required, Validators.maxlength(10)]].

registration-form.component.html

```
<divclass="container">
<h1>Registration Form</h1>
<form [formGroup]="registerForm">
<divclass="form-group">
<label>First Name</label>
<inputtype="text"class="form-
control"formControlName="firstName"/>
<div *ngIf="registerForm.controls.firstName.errors"class="alert
alert-danger">
Firstname field is invalid.
<p *ngIf="registerForm.controls.firstName.errors?.required">
                      This field is required!
 </p>
 </div>
 </div>
 <divclass="form-group">
 <label>Last Name</label>
 <inputtype="text"class="form-
control"formControlName="lastName"/>
 <div *ngIf="registerForm.controls. lastName.errors"class="alert
alert-danger">
                  Lastname field is invalid.
 <p *ngIf="registerForm.controls. lastName.errors?.required">
                      This field is required!
 </p>
 </div>
 </div>
 <divclass="form-group">
 <fieldsetformGroupName="address">
 <label>Street</label>
 <inputtype="text"class="form-control"formControlName="street"/>
 <label>Zip</label>
 <inputtype="text"class="form-control"formControlName="zip"/>
 <label>City</label>
 <inputtype="text"class="form-control"formControlName="city"/>
 </fieldset>
 </div>
 <buttontype="submit"class="btnbtn-primary" (click)="submitted =
true">
        Submit
 </button>
 </form>
 <br/>
 <div [hidden]="!submitted">
 <h3>Employee Details</h3>
```

```
41.  <p>First Name: {{ registerForm.controls.firstName.value }}</p>
42.  <p>Last Name: {{ registerForm.controls.lastName.value }}</p>
43.  <p>Street: {{ registerForm.controls.address.value.street }}</p>
44.  <p>Zip: {{ registerForm.controls.address.value.zip }}</p>
45.  <p>City: {{ registerForm.controls.address.value.city }}</p>
46.  </div>
47.  </div>
48.
```

Line 3: formGroup is a directive that binds HTML form with the FormGroup property created inside a component class. A FormGroup has been created in component with the name registerForm. Here form tag is bound with FormGroup name called registerForm

Line 6, 16: Two text boxes for first name and last name are bound with the form controls created in the component using formControlName directive

The keywords valid, invalid, dirty, etc which has been used in the registration-form.component.html. These indicate the state and validity of a form and a form element.

Line 7-12: A validation error message is displayed when the firstName is modified and has validation errors.

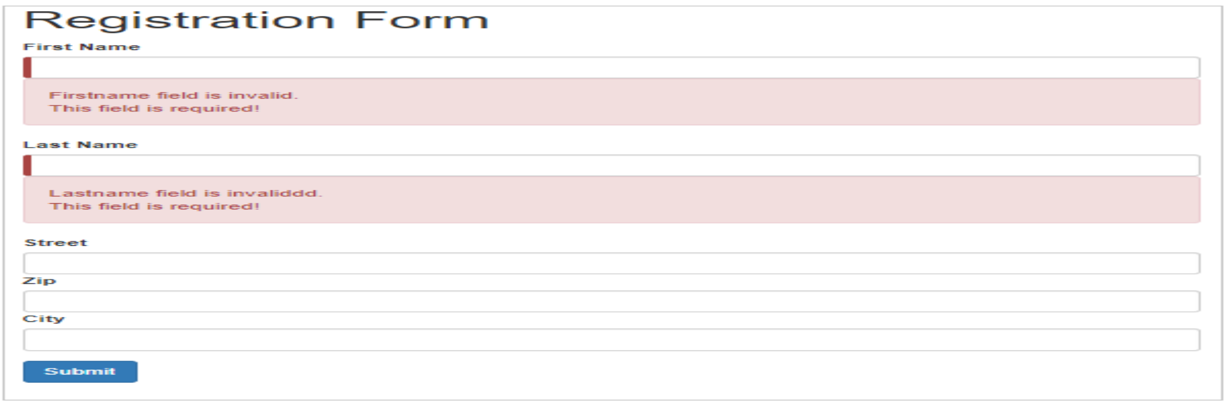
Line 17-22: A validation error message is displayed when the lastName is modified and has validation errors.

Line 34: When the submit button is clicked, it initializes the submitted property value to true

Line 39: div tag will be hidden if the form is not submitted

Line 40-45: Using the get() method of FormGroup, each FormControl value is fetched and rendered.

Save all the files and observe the output:


Registration Form

First Name

Firstname field is invalid.
This field is required!

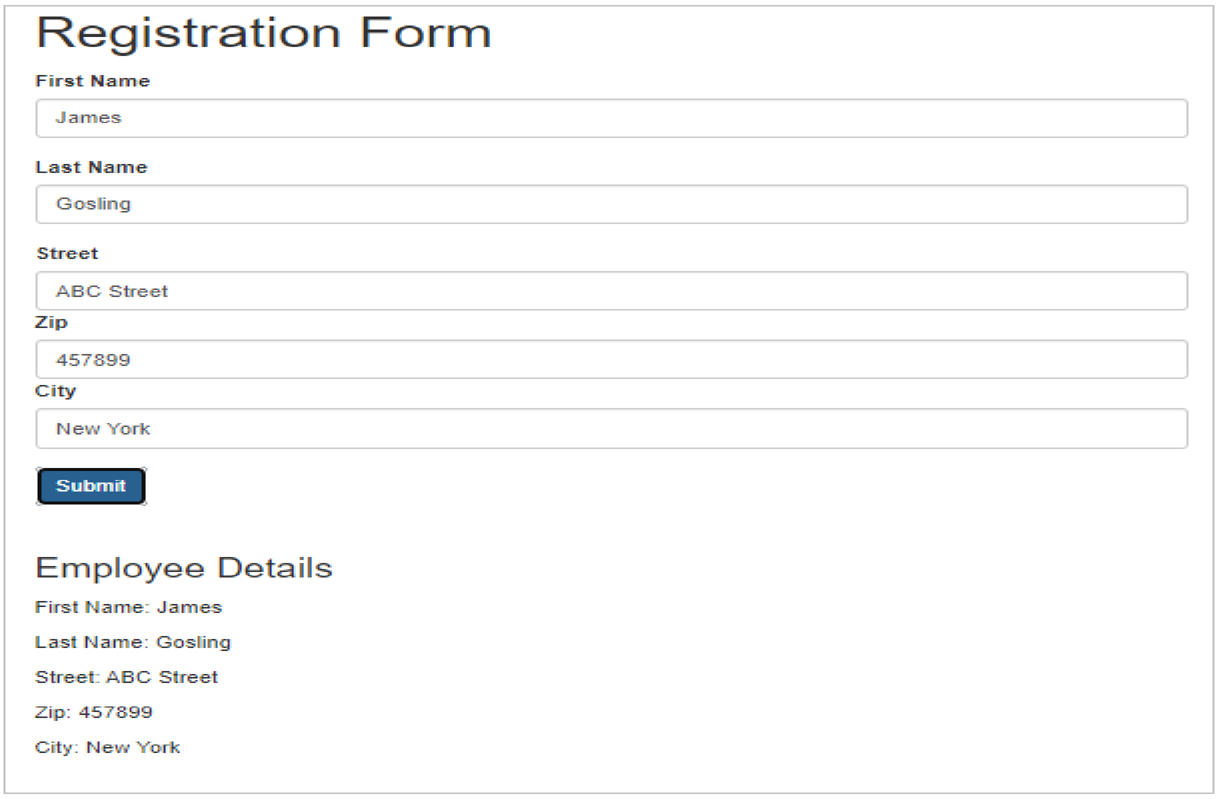
Last Name

Lastname field is invalidddd.
This field is required!

Street

Zip

City


Registration Form

First Name

James

Last Name

Gosling

Street

ABC Street

Zip

457899

City

New York

Employee Details

First Name: James

Last Name: Gosling

Street: ABC Street

Zip: 457899

City: New York

7 B) Course Name: Angular JS
Module Name: Model Driven Forms or Reactive Forms
Create an employee registration form as a reactive form.

1. Write the below-given code in **app.module.ts**

```
import { BrowserModule } from '@angular/platform-browser';

import { NgModule } from '@angular/core';

import { ReactiveFormsModule } from '@angular/forms';

import { AppComponent } from './app.component';

import { RegistrationFormComponent } from './registration-form/registration-form.component';

@NgModule({

  declarations: [

AppComponent,

RegistrationFormComponent

  ],

  imports: [

BrowserModule,

ReactiveFormsModule

  ],

  providers: [],

  bootstrap: [AppComponent]

})

export class AppModule{ }
```

2. Create a component called **RegistrationForm** using the following CLI command

```
ng generate component RegistrationForm
```

3. Add the following code in the **registration-form.component.ts** file

```
import { Component, OnInit } from '@angular/core';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';
@Component({
  selector: 'app-registration-form',
templateUrl: './registration-form.component.html',
styleUrls: ['./registration-form.component.css']
})
export class RegistrationFormComponent implements OnInit {

registerForm!:FormGroup;
submitted!:boolean;

constructor(private formBuilder: FormBuilder) { }

ngOnInit() {
this.registerForm = this.formBuilder.group({
firstName: ['', Validators.required],
lastName: ['', Validators.required],
      address: this.formBuilder.group({
        street: [],
        zip: [],
        city: []
      })
    });
  }
}
```

4. Write the below-given code in **registration-form.component.html**

```
<divclass="container">
<h1>Registration Form</h1>
<form [formGroup]="registerForm">
```

```
<divclass="form-group">
<label>First Name</label>
<inputtype="text"class="form-control"formControlName="firstName">
<div
*ngIf="registerForm.controls['firstName'].errors"class="alert
alert-danger">
Firstname field is invalid.
<p
*ngIf="registerForm.controls['firstName'].errors?.['required']">
                  This field is required!
 </p>
 </div>
 </div>
 <divclass="form-group">
 <label>Last Name</label>
 <inputtype="text"class="form-control"formControlName="lastName">
 <div
*ngIf="registerForm.controls['lastName'].errors"class="alert
alert-danger">
              Lastname field is invalid.
 <p
*ngIf="registerForm.controls['lastName'].errors?.['required']">
                  This field is required!
 </p>
 </div>
 </div>
 <divclass="form-group">
 <fieldsetformGroupName="address">
 <legend>Address:</legend>
 <label>Street</label>
 <inputtype="text"class="form-control"formControlName="street">
 <label>Zip</label>
 <inputtype="text"class="form-control"formControlName="zip">
 <label>City</label>
 <inputtype="text"class="form-control"formControlName="city">
 </fieldset>
 </div>
 <buttontype="submit"class="btnbtn-primary"
(click)="submitted=true">Submit</button>
 </form>
 <br/>
 <div [hidden]="!submitted">
 <h3> Employee Details </h3>
 <p>First Name: {{ registerForm.get('firstName')?.value }} </p>
 <p> Last Name: {{ registerForm.get('lastName')?.value }} </p>
 <p> Street: {{ registerForm.get('address.street')?.value }}</p>
 <p> Zip: {{ registerForm.get('address.zip')?.value }} </p>
 <p> City: {{ registerForm.get('address.city')?.value }}</p>
 </div>
 </div>
```

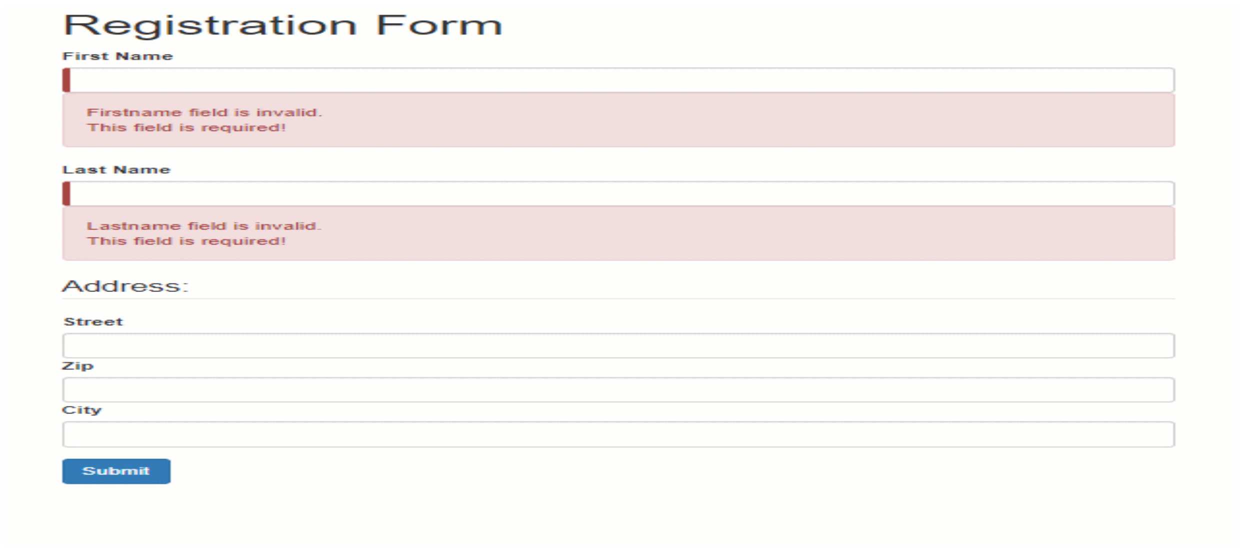
5. Write the below-given code in **registration-form.component.css**

```css
.ng-valid[required]{
  border-left:5px solid #42A948; /* green */
}

.ng-invalid:not(form){
  border-left:5px solid #a94442; /* red */
}

```

6. Write the below-given code in **app.component.html**

```html
<app-registration-form></app-registration-form>
```

7. Save the files and check the output in the browser


Registration Form

First Name

Firstname field is invalid.
This field is required!

Last Name

Lastname field is invalid.
This field is required!

Address:

Street

Zip

City

7 C) Course Name: Angular JS
Module Name: Custom Validators in Reactive Forms
Create a custom validator for an email field in the employee registration form (reactive form)

1. Write a separate function in **registration-form.component.ts** for custom validation as shown below.

```
import{Component,OnInit}from'@angular/core';
import{FormBuilder,FormControl,FormGroup,Validators}from'@angular
/forms';

@Component({
  selector:'app-registration-form',
templateUrl:'./registration-form.component.html',
styleUrls:['./registration-form.component.css']
})
exportclassRegistrationFormComponentimplementsOnInit{

 registerForm!:FormGroup;
 submitted!:boolean;

 constructor(privateformBuilder:FormBuilder){}

 ngOnInit(){
 this.registerForm=this.formBuilder.group({
 firstName:['',Validators.required],
 lastName:['',Validators.required],
       address:this.formBuilder.group({
         street:[],
         zip:[],
         city:[]
 }),
       email:['',[Validators.required,validateEmail]]
 });
 }

 }
 functionvalidateEmail(c:FormControl): any {
 let EMAIL_REGEXP =/^([a-zA-Z0-9_\-\.]+)@([a-zA-Z0-9_\-
\.]+)\.([a-zA-Z]{2,5})$/;

 returnEMAIL_REGEXP.test(c.value)?null:{
 emailInvalid:{
       message:"Invalid Format!"
 }
 };

 }
```

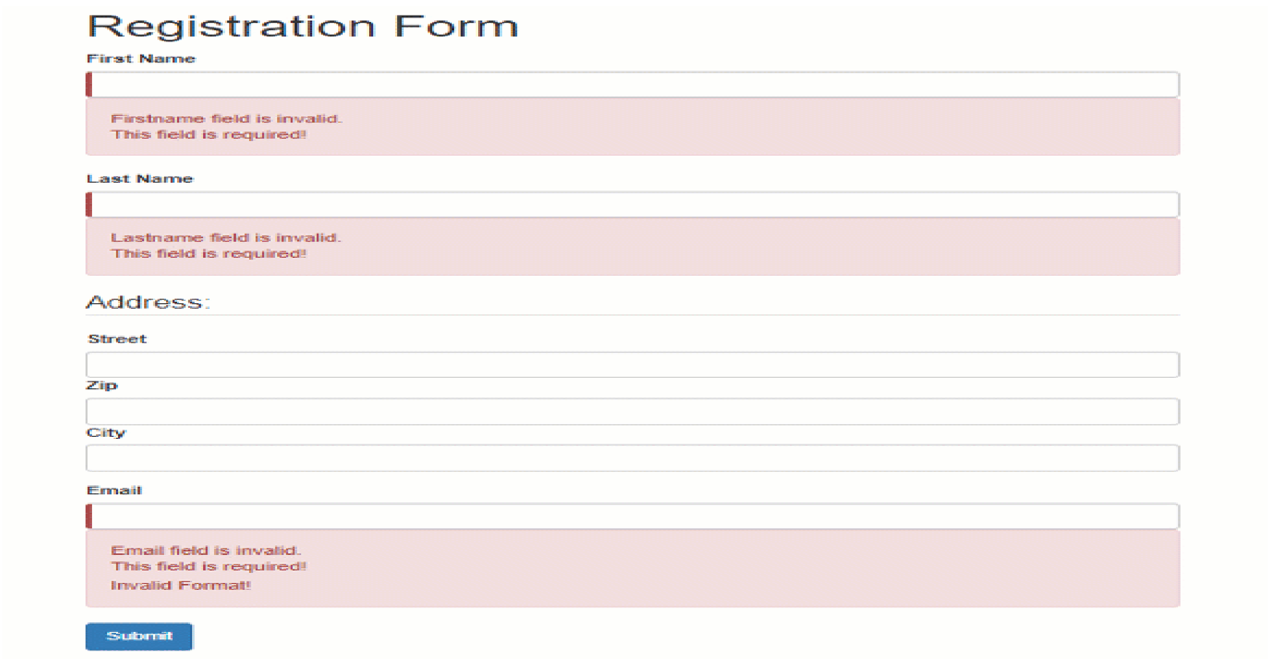
2. Add HTML controls for the email field in the **registration-form.component.html** file as shown below

```
<divclass="container">
<h1>Registration Form</h1>
<form [formGroup]="registerForm">
<divclass="form-group">
```

```
<label>First Name</label>
<inputtype="text"class="form-control"formControlName="firstName">
<div
*ngIf="registerForm.controls['firstName'].errors"class="alert
alert-danger">
Firstname field is invalid.
<p
*ngIf="registerForm.controls['firstName'].errors?.['required']">
                 This field is required!
 </p>
 </div>
 </div>
 <divclass="form-group">
 <label>Last Name</label>
 <inputtype="text"class="form-control"formControlName="lastName">
 <div
*ngIf="registerForm.controls['lastName'].errors"class="alert
alert-danger">
             Lastname field is invalid.
 <p
*ngIf="registerForm.controls['lastName'].errors?.['required']">
                 This field is required!
 </p>
 </div>
 </div>
 <divclass="form-group">
 <fieldsetformGroupName="address">
 <legend>Address:</legend>
 <label>Street</label>
 <inputtype="text"class="form-control"formControlName="street">
 <label>Zip</label>
 <inputtype="text"class="form-control"formControlName="zip">
 <label>City</label>
 <inputtype="text"class="form-control"formControlName="city">
 </fieldset>
 </div>
 <divclass="form-group">
 <label>Email</label>
 <inputtype="text"class="form-control"formControlName="email"/>
 <div *ngIf="registerForm.controls['email'].errors"class="alert
alert-danger">
           Email field is invalid.
 <p *ngIf="registerForm.controls['email'].errors?.['required']">
             This field is required!
 </p>
 <p
*ngIf="registerForm.controls['email'].errors?.['emailInvalid']">
 {{
registerForm.controls['email'].errors?.['emailInvalid'].message
}}
 </p>
 </div>
 </div>
 <buttontype="submit"class="btnbtn-primary"
(click)="submitted=true">Submit</button>
 </form>
 <br/>
 <div [hidden]="!submitted">
```

```
<h3> Employee Details </h3>
<p>First Name: {{ registerForm.get('firstName')?.value }} </p>
<p> Last Name: {{ registerForm.get('lastName')?.value }} </p>
<p> Street: {{ registerForm.get('address.street')?.value }}</p>
<p> Zip: {{ registerForm.get('address.zip')?.value }} </p>
<p> City: {{ registerForm.get('address.city')?.value }}</p>
<p>Email: {{ registerForm.get('email')?.value }}</p>
</div>
</div>



```

3. Save the files and check the output in the browser


Registration Form

First Name

Firstname field is invalid.
This field is required!

Last Name

Lastname field is invalid.
This field is required!

Address:

Street

Zip

City

Email

Email field is invalid.
This field is required!
Invalid Format!

8 A) Course Name: Angular JS
Module Name: Custom Validators in Template Driven forms
Create a custom validator for the email field in the course registration form.

- The code inside **login.component.html** is present under the login folder. Observe the creation of LoginComponent as a reactive form and addition of validations to it.

```
<!-- Login form-->
<divclass="container container-styles">
  <divclass="col-xs-7 col-xs-offset-3">
        <divclass="panel panel-primary">
                <divclass="panel-heading">Login</div>
                <divclass="panel-body padding">
```

```
                        <formclass="form-horizontal" [formGroup]="loginForm">
                                <divclass="form-group">
                                        <labelfor="name"class="col-xs-4 control-label"style="text-align:left">User Name</label>
                                                <divclass="col-xs-8">
<inputtype="text"class="form-control" [ngClass]="{'valid':loginForm.controls['userName'].valid, 'invalid':loginForm.controls['userName'].invalid && !loginForm.controls['userName'].pristine}"formControlName="userName">
                                                        <div *ngIf="loginForm.controls['password'].errors&&loginForm.controls['userName'].dirty">
                                                        <div *ngIf="loginForm.controls['userName'].errors?.['required']"style="color:red">UserName is required
                                                        </div>
                                                </div>
                                                </div>
                                        </div>
                                        <divclass="form-group">
<labelfor="password"class="col-xs-4 control-label"style="text-align:left">Password</label>
                                                <divclass="col-xs-8">
<inputtype="password"class="form-control" [ngClass]="{'valid':loginForm.controls['password'].valid, 'invalid':loginForm.controls['password'].invalid && !loginForm.controls['password'].pristine}"formControlName="password">
                                                        <div *ngIf="loginForm.controls['password'].errors&&loginForm.controls['password'].dirty">
                                                        <div *ngIf="loginForm.controls['password'].errors?.['required']"style="color:red">Password is required
                                                        </div>
                                                </div>
                                                </div>
                                        </div>

                                        <div *ngIf="!valid"class="error">Invalid Credentials...Please try again...</div>
                                        <br/>

                                        <divclass="form-group">
                                                <spanclass="col-xs-4"></span>
                                                <divclass="col-xs-3">
```

```
35.                                                     <button
   (click)="onSubmit()"class="btnbtn-primary"
   [disabled]="!loginForm.valid">Login</button>
36.                                              </div>
37.                                              <spanclass="col-xs-
   5"style="top:8px">
38.                                                     <a
   [routerLink]="['/welcome']"style="color:#337ab7;text-decoration:
   underline;">Cancel</a>
39.                                              </span>
40.                                       </div>
41.                                </form>
42.                         </div>
43.                  </div>
44.           </div>
45.    </div>


46.
```

Line 11: The username textbox is bound to the formControlNameuserName

Line 21: The password textbox is bound to the formControlName password

Line 12-16: Renders the given error message if required validation fails

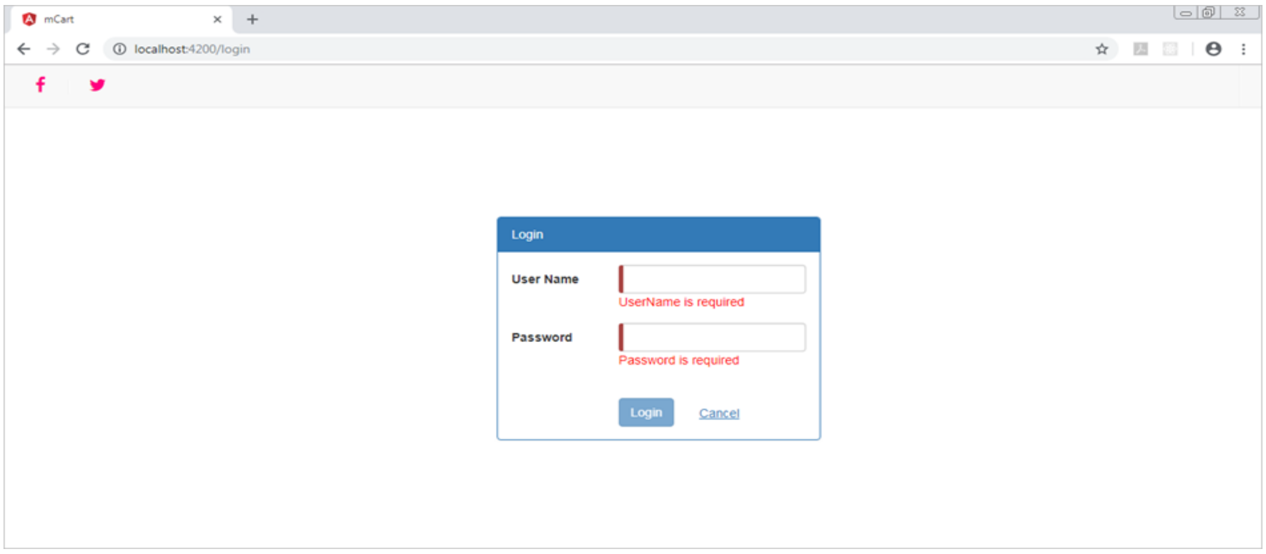
Line 22-26: Similarly required validation is added to the password field and the corresponding error message will be displayed if the validation fails

Observe the creation of model-driven form within the LoginComponent. Open login.component.ts.

```
1. import{Component,ElementRef,OnInit,Renderer2,ViewChild}from'@angu
   lar/core';
2. import{FormBuilder,FormGroup,Validators}from'@angular/forms';
3. import{Router}from'@angular/router';
4. import{Login}from'./Login';
5. import{LoginService}from'./login.service';
6.
7. @Component({
8. templateUrl:'./login.component.html',
9. styleUrls:['./login.component.css']
10.  })
11.  exportclassLoginComponentimplementsOnInit{
12.
13.      login =newLogin();
14.      users:Login[]=[];
15.      valid =true;
16.  @ViewChild('uname')usernameElement!:ElementRef;
17.  loginForm!:FormGroup;
18.
19.  constructor(private
   router:Router,privateformBuilder:FormBuilder,
20.  privateloginService:LoginService,private renderer:Renderer2){
21.  }
22.
23.  ngOnInit(){
24.  // Makes a service call to fetch users data from the backend
25.  this.loginService.getUsers().subscribe({next:users=>this.users=
   users});
26.  this.loginForm=this.formBuilder.group({
27.  userName:[this.login.userName,Validators.required],
28.               password:[this.login.password,Validators.required]
29.  })
30.  }
31.
32.  // Invoked when user clicks submit in login form
33.  // Validates the credentials with the data fetched from the
   backend
34.  onSubmit(){
35.
36.  //fetches the form object containing the values of all the form
   controls
37.  this.login=this.loginForm.getRawValue();
38.  const user
   =this.users.filter(currUser=>currUser.userName===this.login.userN
   ame&&currUser.password===this.login.password)[0];
39.  if(user){
40.  this.loginService.username=this.login.userName;
41.  this.router.navigate(['/products']);
42.  }else{
43.  this.valid=false;
44.  }
45.  }
46.  }

47.
```

Line 26-29: Builds the form with two form input controls: userName, password

OUTPUT:-



Login

User Name

UserName is required

Password

Password is required

Login Cancel

8 B)Course Name: Angular JS
Module Name: Services Basics

Create a Book Component which fetches book details like id, name and displays them on the page in a list format. Store the book details in an array and fetch the data using a custom service.

Problem Statement: Create a Book Component which fetches book details like id, name and displays them on the page in a list format. Store the book details in an array and fetch the data using a custom service. The output is as shown below

1. Create **BookComponent** by using the following CLI command

```
D:\MyApp>ng generate component book
```

2. Create a file with the name **book.ts** under the book folder and add the following code.

```
exportclassBook {
id!: number;
name!: string;
}

```

3. Create a file with the name **books-data.ts** under the book folder and add the following code.

```
import{ Book } from'./book';

exportlet BOOKS: Book[] = [
{ id: 1, name: 'HTML 5' },
{ id: 2, name: 'CSS 3' },
{ id: 3, name: 'Java Script' },
{ id: 4, name: 'Ajax Programming' },
{ id: 5, name: 'jQuery' },
{ id: 6, name: 'Mastering Node.js' },
 { id: 7, name: 'Angular JS 1.x' },
 { id: 8, name: 'ng-book 2' },
 { id: 9, name: 'Backbone JS' },
 { id: 10, name: 'Yeoman' }
 ];

```

4. Create a service called **BookService** under the book folder using the following CLI command

```
D:\MyApp\src\app\book>ng generate service book
```

5. Add the following code in **book.service.ts**

```
import{ Injectable } from'@angular/core';
import{ BOOKS } from'./books-data';

@Injectable({
providedIn: 'root'
})

exportclassBookService {
getBooks() {
 return BOOKS;
   }
 }

```

6. Add the following code in the **book.component.ts** file

```
import{ Component, OnInit } from'@angular/core';
import{ Book } from'./book';
import{ BookService } from'./book.service';

@Component({
  selector: 'app-book',
templateUrl: './book.component.html',
styleUrls: ['./book.component.css']
})
 exportclassBookComponentimplementsOnInit {

 books!:Book[];

 constructor(privatebookService: BookService) { }
 getBooks() {
 this.books = this.bookService.getBooks();
   }
 ngOnInit() {
 this.getBooks();
   }
 }
```

```

```

7. Write the below-given code in **book.component.html**

```html
<h2>My Books</h2>
<ulclass="books">
<li *ngFor="let book of books">
<spanclass="badge">{{book.id}}</span> {{book.name}}
</li>

</ul>
```

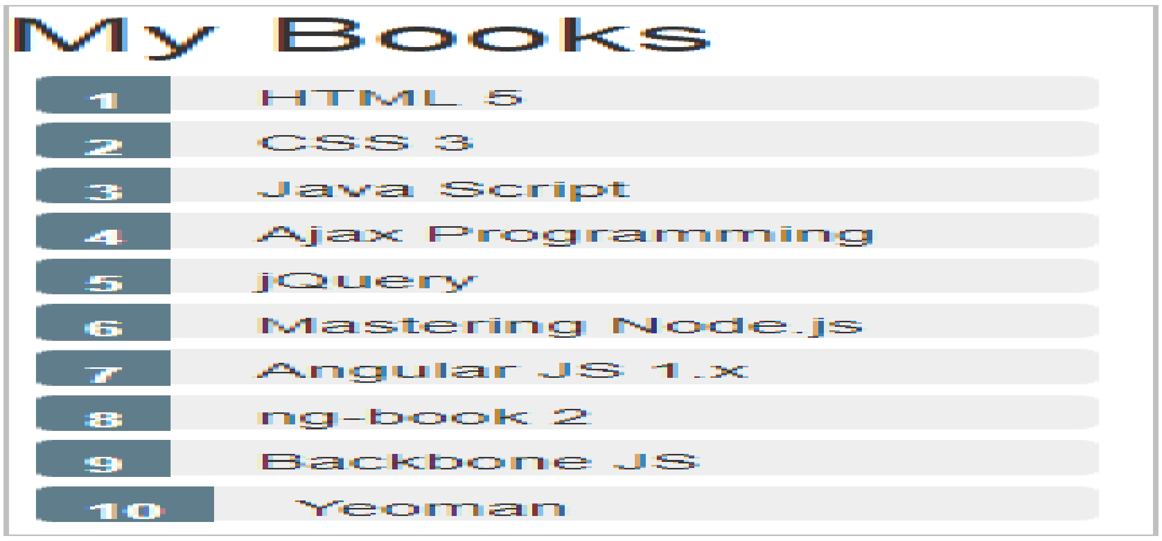
8. Add the following code in **book.component.css** which has styles for books

```css
.books {
  margin: 002em0;
  list-style-type: none;
  padding: 0;
  width: 13em;
}
.books li {
  cursor: pointer;
  position: relative;
    left: 0;
    background-color: #eee;
    margin: 0.5em;
    padding: 0.3em0;
    height: 1.5em;
    border-radius: 4px;
  }
  .booksli:hover {
    color: #607d8b;
    background-color: #ddd;
    left: 0.1em;
  }
  .books .badge {
    display: inline-block;
    font-size: small;
    color: white;
    padding: 0.8em0.7em00.7em;
    background-color: #607d8b;
    line-height: 0.5em;
    position: relative;
    left: -1px;
    top: -4px;
    height: 1.8em;
    margin-right: 0.8em;
    border-radius: 4px004px;
}
```

9. Add the following code in app.component.html

```
<app-book></app-book>
```

10. **Save the files and check the output in the browser**



```
My Books
1   HTML 5
2   CSS 3
3   Java Script
4   Ajax Programming
5   jQuery
6   Mastering Node.js
7   Angular JS 1.x
8   ng-book 2
9   Backbone JS
10  Yeoman
```

8 C) Course Name: Angular JS
Module Name: RxJS Observables
Create and use an observable in Angular

PROGRAM:-

RxJS

Reactive Extensions for JavaScript (RxJS) is a third-party library used by the Angular team.
RxJS is a reactive streams library used to work with asynchronous streams of data.
Observables, in RxJS, are used to represent asynchronous streams of data. Observables are a more advanced version of Promises in JavaScript

Why RxJS Observables?

Angular team has recommended Observables for asynchronous calls because of the following reasons:

1. Promises emit a single value whereas observables (streams) emit many values
2. Observables can be cancellable where Promises are not cancellable. If an HTTP response is not required, observables allow us to cancel the subscription whereas promises execute either success or failure callback even if the results are not required.
3. Observables support functional operators such as map, filter, reduce, etc.,

Create and use an observable in Angular

Example:**app.component.ts**

```
import{ Component } from'@angular/core';
import{ Observable } from'rxjs';

@Component({
  selector: 'app-root',
styleUrls: ['./app.component.css'],
templateUrl: './app.component.html'
})
exportclassAppComponent {

 data!:Observable<number>;
 myArray: number[] = [];
 errors!:boolean;
 finished!:boolean;

 fetchData(): void {
 this.data = newObservable(observer => {
 setTimeout(() => { observer.next(11); }, 1000),
 setTimeout(() => { observer.next(22); }, 2000),
 setTimeout(() => { observer.complete(); }, 3000);
     });
 this.data.subscribe((value) =>this.myArray.push(value),
       error =>this.errors = true,
       () =>this.finished = true);
   }
 }
```

```
27.
```

Line 2: imports Observable class from rxjs module

Line 11: data is of type Observable which holds numeric values

Line 16: fetchData() is invoked on click of a button

Line 17: A new Observable is created and stored in the variable data

Line 18-20: next() method of Observable sends the given data through the stream. With a delay of 1,2 and 3 seconds, a stream of numeric values will be sent. Complete() method completes the Observable stream i.e., closes the stream.

Line 22: Observable has another method called subscribe which listens to the data coming through the stream. Subscribe() method has three parameters. The first parameter is a success callback which will be invoked upon receiving successful data from the stream. The second parameter is an error callback which will be invoked when Observable returns an error and the third parameter is a complete callback which will be invoked upon successful streaming of values from Observable i.e., once complete() is invoked. After which the successful response, the data is pushed to the local array called myArray, if any error occurs, a Boolean value called true is stored in the errors variable and upon complete() will assign a Boolean value true in a finished variable.

app.component.html

```
1. <b> Using Observables!</b>
2.
3. <h6style="margin-bottom: 0">VALUES:</h6>
4. <div *ngFor="let value of myArray">{{ value }}</div>
5.
6. <divstyle="margin-bottom: 0">ERRORS: {{ errors }}</div>
7.
8. <divstyle="margin-bottom: 0">FINISHED: {{ finished }}</div>
9.
10.  <buttonstyle="margin-top: 2rem"  (click)="fetchData()">Fetch
   Data</button>

11.
```

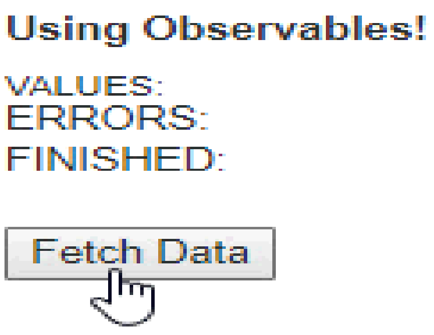
Line 4: ngFor loop is iterated on myArray which will display the values on the page

Line 6: {{ errors }} will render the value of errors property if any

Line 8: Displays finished property value when complete() method of Observable is executed

Line 10: Button click event is bound with fetchData() method which is invoked and creates an observable with a stream of numeric values

Output:

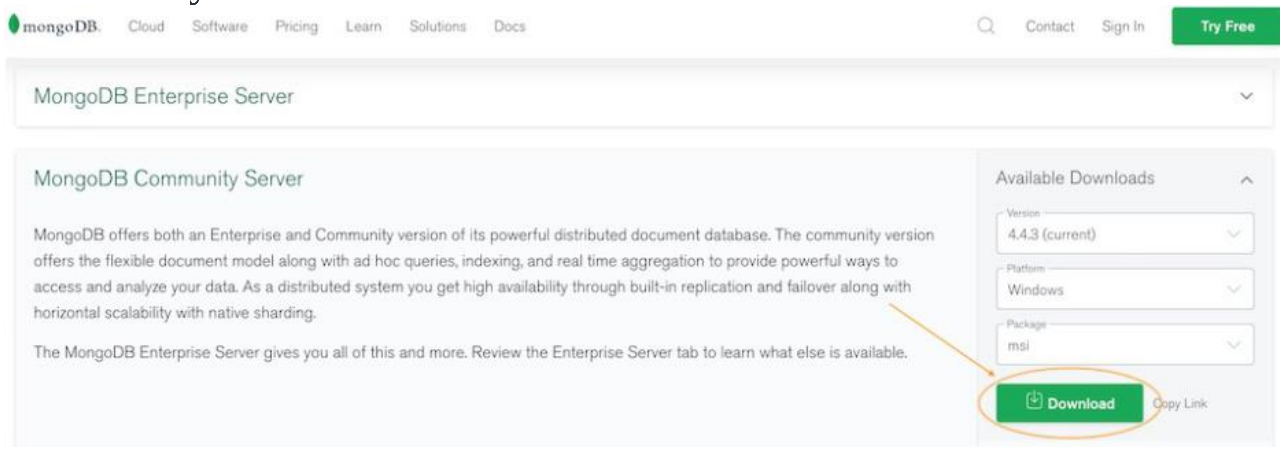

Using Observables!

VALUES:
ERRORS:
FINISHED:

11a): Install MongoDB and configure ATLAS

How to Install MongoDB on Windows?

Step 1: Go to MongoDB Download Center to download MongoDB Community Server.



MongoDB Enterprise Server

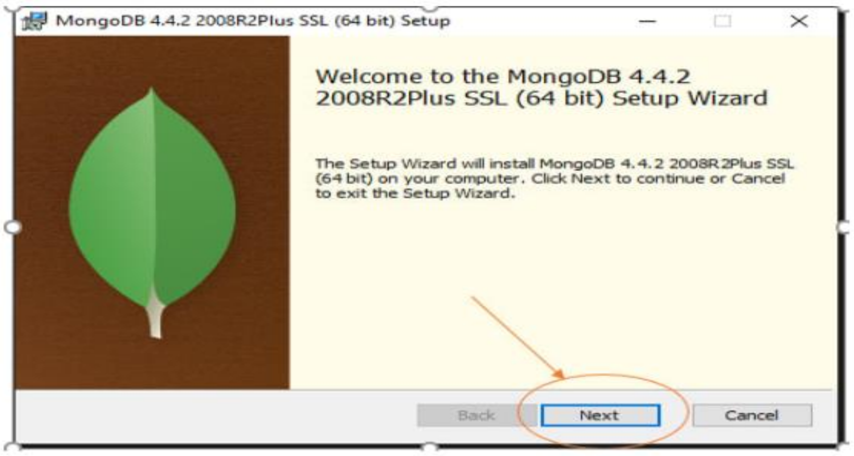
MongoDB Community Server

MongoDB offers both an Enterprise and Community version of its powerful distributed document database. The community version offers the flexible document model along with ad hoc queries, indexing, and real time aggregation to provide powerful ways to access and analyze your data. As a distributed system you get high availability through built-in replication and failover along with horizontal scalability with native sharding.

The MongoDB Enterprise Server gives you all of this and more. Review the Enterprise Server tab to learn what else is available.

Version: 4.4.3 (current)
Platform: Windows
Package: msi

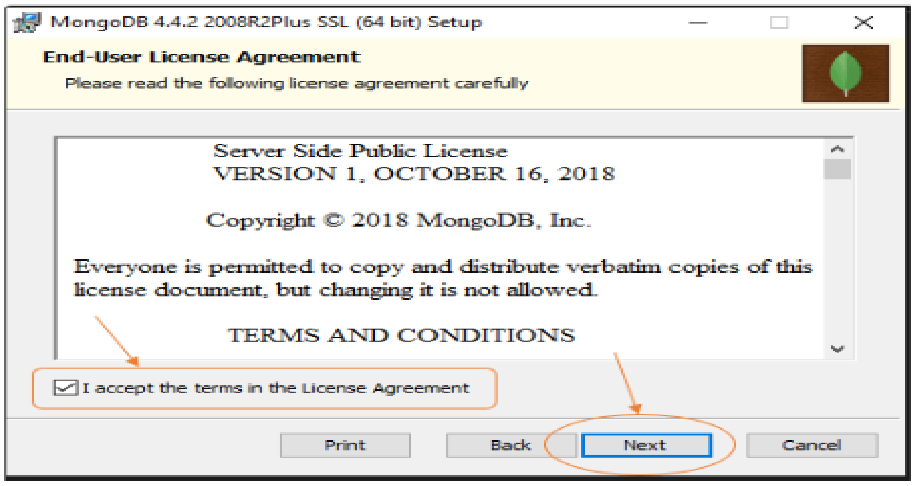
Step 2: When the download is complete open the msi file and click the *next button* in the startup screen:


MongoDB 4.4.2 2008R2Plus SSL (64 bit) Setup

Welcome to the MongoDB 4.4.2 2008R2Plus SSL (64 bit) Setup Wizard

The Setup Wizard will install MongoDB 4.4.2 2008R2Plus SSL (64 bit) on your computer. Click Next to continue or Cancel to exit the Setup Wizard.

Step 3: Now accept the End-User License Agreement and click the next buttc



MongoDB 4.4.2 2008R2Plus SSL (64 bit) Setup

End-User License Agreement

Please read the following license agreement carefully

Server Side Public License
VERSION 1, OCTOBER 16, 2018

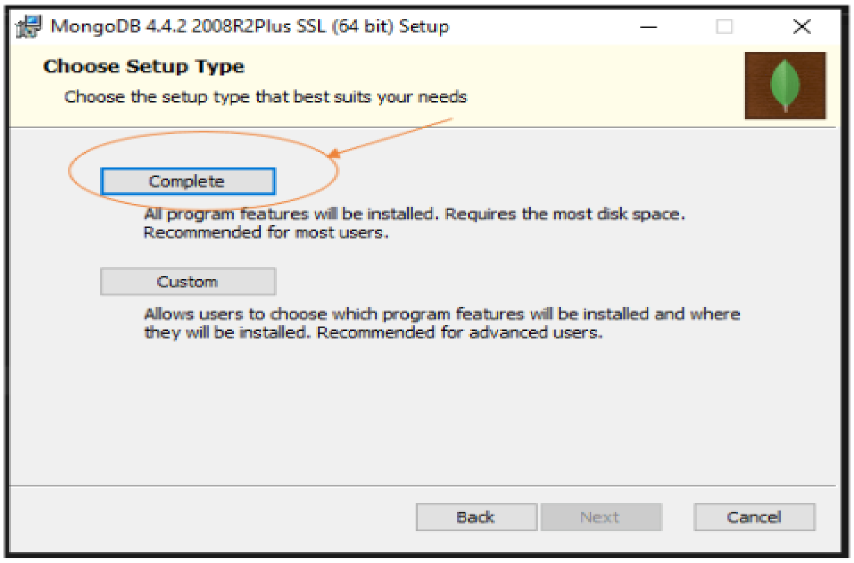
Copyright © 2018 MongoDB, Inc.

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

TERMS AND CONDITIONS

☑ I accept the terms in the License Agreement

Step 4: Now select the *complete option* to install all the program features. Here, if you can want to install only selected program features and want to select the location of the installation, then use the *Custom option:*



MongoDB 4.4.2 2008R2Plus SSL (64 bit) Setup

Choose Setup Type

Choose the setup type that best suits your needs

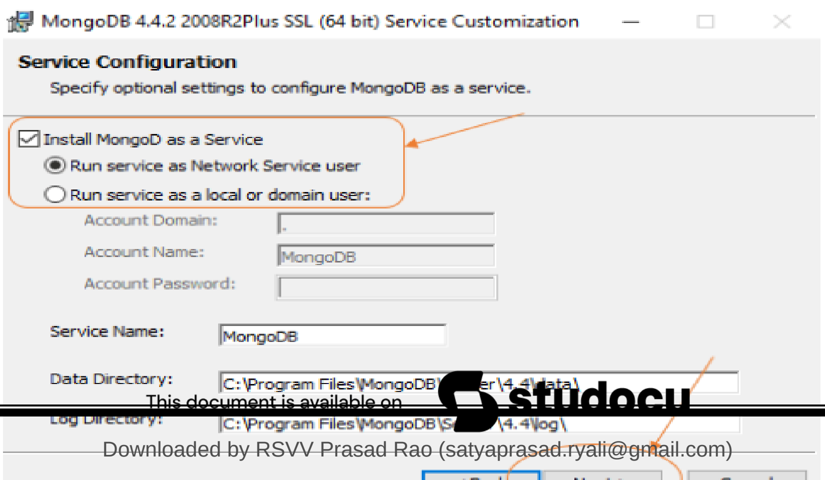
Complete

All program features will be installed. Requires the most disk space. Recommended for most users.

Custom

Allows users to choose which program features will be installed and where they will be installed. Recommended for advanced users.

Step 5: Select "Run service as Network Service user" and copy the path of the data directory. Click Next:



MongoDB 4.4.2 2008R2Plus SSL (64 bit) Service Customization

Service Configuration

Specify optional settings to configure MongoDB as a service.

☑ Install MongoD as a Service

● Run service as Network Service user

○ Run service as a local or domain user:

Account Domain: .

Account Name: MongoDB

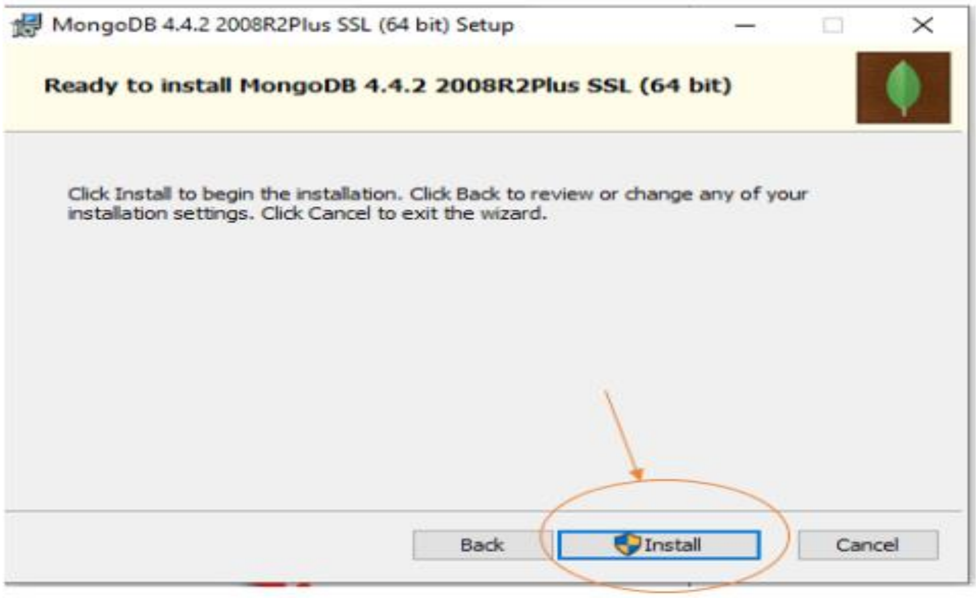
Account Password:

Service Name: MongoDB

Data Directory: C:\Program Files\MongoDB\ [illegible] er\4.4\data\

Log Directory: C:\Program Files\MongoDB\S [illegible] \4.4\log\

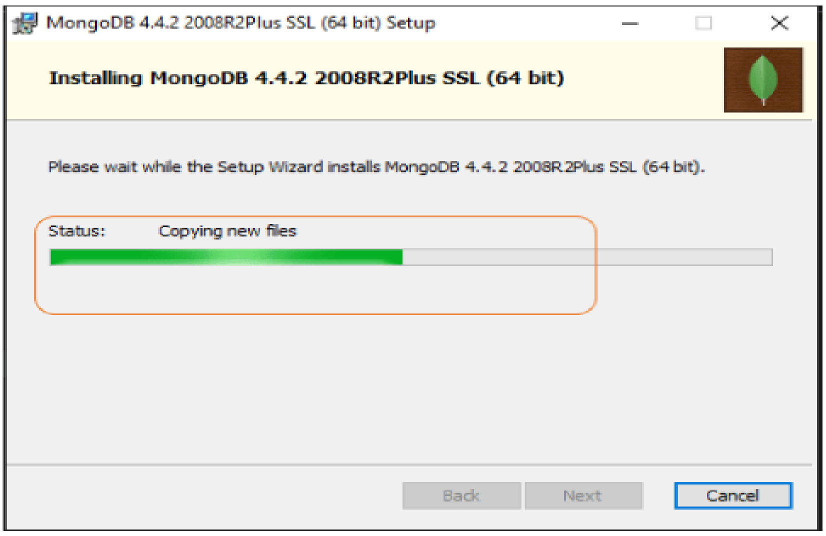
Step 6: Click the *Install button* to start the installation process:


MongoDB 4.4.2 2008R2Plus SSL (64 bit) Setup

Ready to install MongoDB 4.4.2 2008R2Plus SSL (64 bit)

Click Install to begin the installation. Click Back to review or change any of your installation settings. Click Cancel to exit the wizard.

Step 7: After clicking on the install button installation of MongoDB begins:

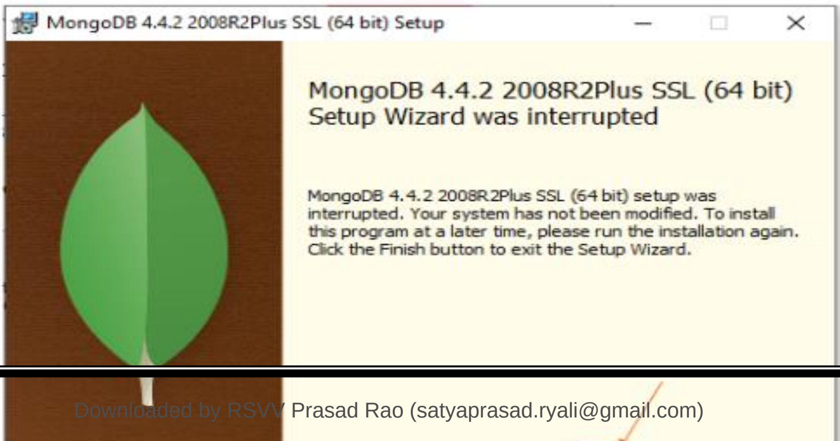

MongoDB 4.4.2 2008R2Plus SSL (64 bit) Setup

Installing MongoDB 4.4.2 2008R2Plus SSL (64 bit)

Please wait while the Setup Wizard installs MongoDB 4.4.2 2008R2Plus SSL (64 bit).

Status: Copying new files

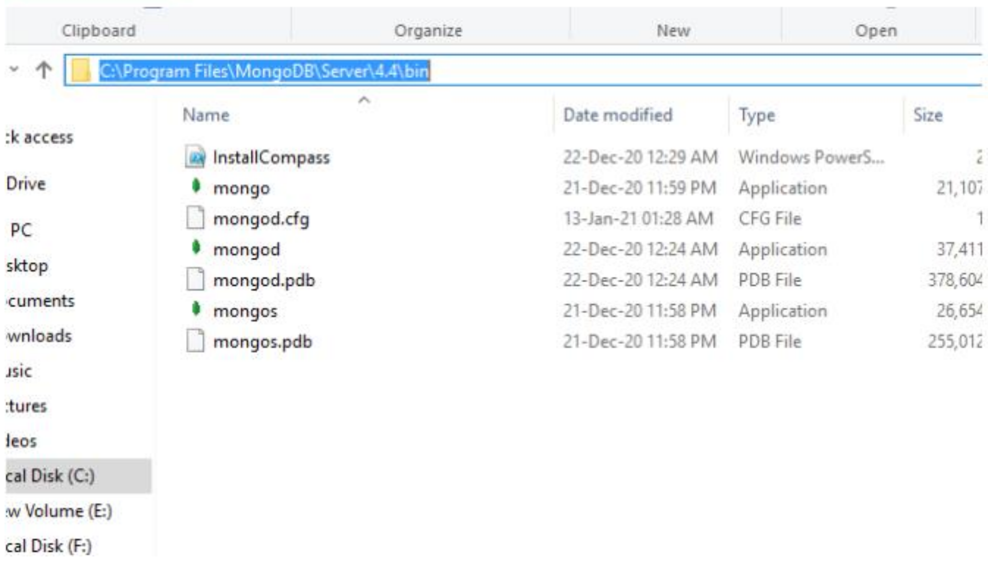
Step 8: Now click the *Finish button* to complete the installation process


MongoDB 4.4.2 2008R2Plus SSL (64 bit) Setup

MongoDB 4.4.2 2008R2Plus SSL (64 bit) Setup Wizard was interrupted

MongoDB 4.4.2 2008R2Plus SSL (64 bit) setup was interrupted. Your system has not been modified. To install this program at a later time, please run the installation again. Click the Finish button to exit the Setup Wizard.

Step 9: Now we go to the location where MongoDB installed in step 5 in your system and copy the bin path:

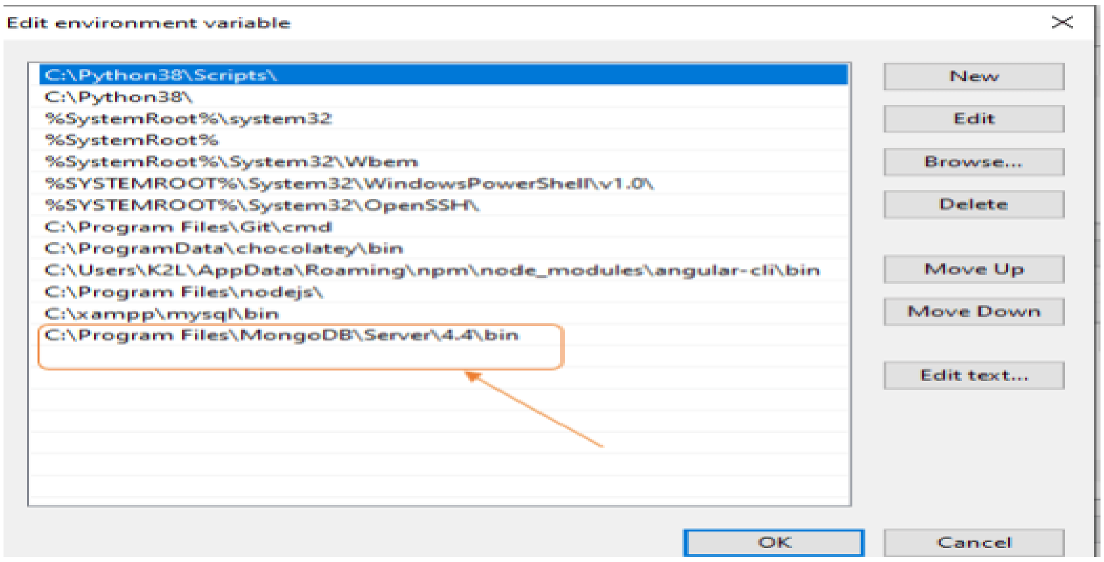


Clipboard Organize New Open

C:\Program Files\MongoDB\Server\4.4\bin

Name	Date modified	Type	Size
InstallCompass	22-Dec-20 12:29 AM	Windows PowerS...	2
mongo	21-Dec-20 11:59 PM	Application	21,107
mongod.cfg	13-Jan-21 01:28 AM	CFG File	1
mongod	22-Dec-20 12:24 AM	Application	37,411
mongod.pdb	22-Dec-20 12:24 AM	PDB File	378,604
mongos	21-Dec-20 11:58 PM	Application	26,654
mongos.pdb	21-Dec-20 11:58 PM	PDB File	255,012

Step 10: Now, to create an environment variable open system properties<< Environment Variable << System variable << path << Edit Environment variable and paste the copied link to your environment system and click Ok:



Edit environment variable

```
C:\Python38\Scripts\
C:\Python38\
%SystemRoot%\system32
%SystemRoot%
%SystemRoot%\System32\Wbem
%SYSTEMROOT%\System32\WindowsPowerShell\v1.0\
%SYSTEMROOT%\System32\OpenSSH\
C:\Program Files\Git\cmd
C:\ProgramData\chocolatey\bin
C:\Users\K2L\AppData\Roaming\npm\node_modules\angular-cli\bin
C:\Program Files\nodejs\
C:\xampp\mysql\bin
C:\Program Files\MongoDB\Server\4.4\bin
```

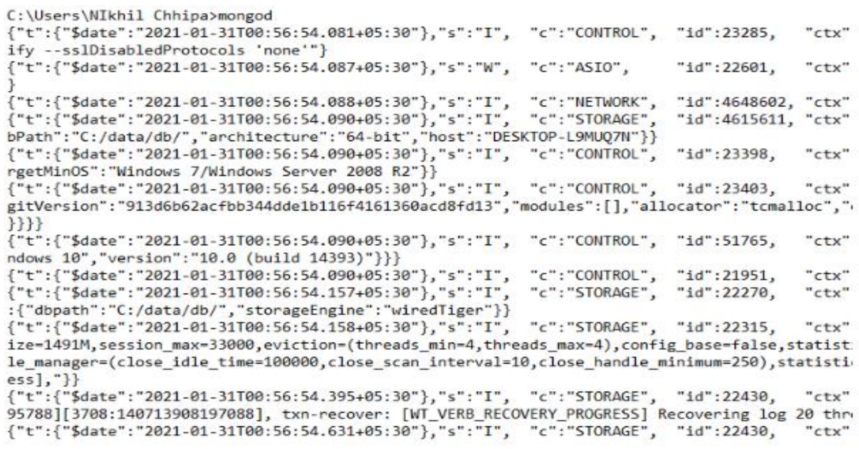
Step 11: After setting the environment variable, we will run the MongoDB server, i.e.mongod. So, open the command prompt and run the following command:

mongod

Step 12: Now, Open C drive and create a folder named “data” inside this folder create another folder named “db”. After creating these folders. Again open the command prompt and run the following command:

mongod

Now, this time the MongoDB server(i.e., mongod) will run successfully.

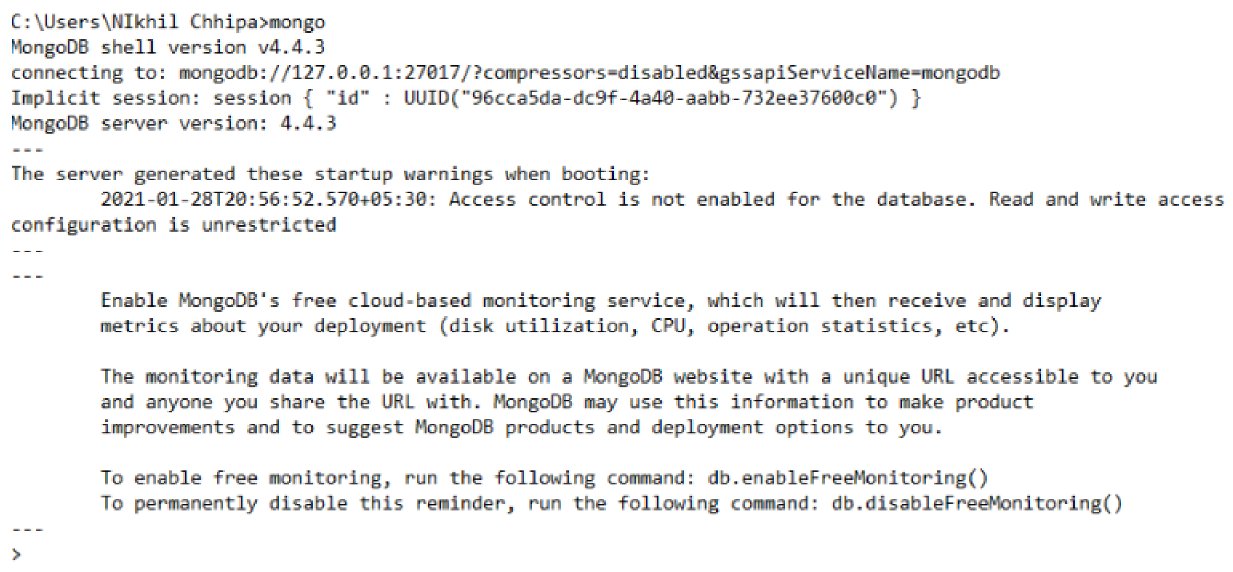


```
C:\Users\NIkhil Chhipa>mongod
{"t":{"$date":"2021-01-31T00:56:54.081+05:30"},"s":"I",  "c":"CONTROL",  "id":23285,   "ctx"
ify --sslDisabledProtocols 'none'"}
{"t":{"$date":"2021-01-31T00:56:54.087+05:30"},"s":"W",  "c":"ASIO",     "id":22601,   "ctx"
}
{"t":{"$date":"2021-01-31T00:56:54.088+05:30"},"s":"I",  "c":"NETWORK",  "id":4648602, "ctx"
{"t":{"$date":"2021-01-31T00:56:54.090+05:30"},"s":"I",  "c":"STORAGE",  "id":4615611, "ctx"
bPath":"C:/data/db/","architecture":"64-bit","host":"DESKTOP-L9MUQ7N"}}
{"t":{"$date":"2021-01-31T00:56:54.090+05:30"},"s":"I",  "c":"CONTROL",  "id":23398,   "ctx"
rgetMinOS":"Windows 7/Windows Server 2008 R2"}}
{"t":{"$date":"2021-01-31T00:56:54.090+05:30"},"s":"I",  "c":"CONTROL",  "id":23403,   "ctx"
gitVersion":"913d6b62acfbb344dde1b116f4161360acd8fd13","modules":[],"allocator":"tcmalloc","
}}}}
{"t":{"$date":"2021-01-31T00:56:54.090+05:30"},"s":"I",  "c":"CONTROL",  "id":51765,   "ctx"
ndows 10","version":"10.0 (build 14393)"}}}
{"t":{"$date":"2021-01-31T00:56:54.090+05:30"},"s":"I",  "c":"CONTROL",  "id":21951,   "ctx"
{"t":{"$date":"2021-01-31T00:56:54.157+05:30"},"s":"I",  "c":"STORAGE",  "id":22270,   "ctx"
:{"dbpath":"C:/data/db/","storageEngine":"wiredTiger"}}
{"t":{"$date":"2021-01-31T00:56:54.158+05:30"},"s":"I",  "c":"STORAGE",  "id":22315,   "ctx"
ize=1491M,session_max=33000,eviction=(threads_min=4,threads_max=4),config_base=false,statist
le_manager=(close_idle_time=100000,close_scan_interval=10,close_handle_minimum=250),statisti
ess],"}}
{"t":{"$date":"2021-01-31T00:56:54.395+05:30"},"s":"I",  "c":"STORAGE",  "id":22430,   "ctx"
95788][3708:140713908197088], txn-recover: [WT_VERB_RECOVERY_PROGRESS] Recovering log 20 thr
{"t":{"$date":"2021-01-31T00:56:54.631+05:30"},"s":"I",  "c":"STORAGE",  "id":22430,   "ctx"
```

Run mongo Shell

Step 13: Now we are going to connect our server (mongod) with the mongo shell. So, keep that mongod window and open a new command prompt window and write **mongo.** Now, our mongo shell will successfully connect to the mongod.

Important Point: Please do not close the mongod window if you close this window your server will stop working and it will not able to connect with the mongo shell.



```
C:\Users\NIkhil Chhipa>mongo
MongoDB shell version v4.4.3
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("96cca5da-dc9f-4a40-aabb-732ee37600c0") }
MongoDB server version: 4.4.3
---
The server generated these startup warnings when booting:
        2021-01-28T20:56:52.570+05:30: Access control is not enabled for the database. Read and write access
configuration is unrestricted
---
---
        Enable MongoDB's free cloud-based monitoring service, which will then receive and display
        metrics about your deployment (disk utilization, CPU, operation statistics, etc).

        The monitoring data will be available on a MongoDB website with a unique URL accessible to you
        and anyone you share the URL with. MongoDB may use this information to make product
        improvements and to suggest MongoDB products and deployment options to you.

        To enable free monitoring, run the following command: db.enableFreeMonitoring()
        To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---
>
```

Now, you are ready to write queries in the mongo Shell.

Mongodb ATLAS Cluster: MongoDB Atlas provides an easy way to host and manage your data in the cloud.

You can get started with Atlas through the Atlas CLI or the Atlas User Interface. Select a tab based on how you would like to get started.

To create and authenticate with your Atlas account, create one free database, load sample data, add your IP address to your project IP access list, create a MongoDB user, and view your connection string using the Atlas CLI, run the following command.

atlas setup [options]

Create an Atlas Account: Register for an Atlas account using your GitHub account, your Google account or your email address.

Register a new Atlas Account

Select a tab based on how you would like to register your account.

To register with Atlas using the Atlas CLI, run the following command:

```
atlas auth register [options]
```

Log in to Your Atlas Account: To authenticate with Atlas using the Atlas CLI, run the following command:

```
atlas auth login [options]
```

Create an Atlas Organization and Project

Create an Atlas organization and then create a project in this organization. You will deploy your first cluster in this project.

Deploy a Free Cluster

Atlas free clusters provide a small-scale development environment to host your data. Free clusters never expire, and provide access to a subset of Atlas features and functionality.

Procedure:

You can create free clusters through the Atlas CLI, Atlas User Interface, and Atlas Administration API. Select the appropriate tab based on how you would like to create the free clusters.

To create one cluster, load sample data, add your IP address to your project IP access list, and create a MongoDB user for your cluster using the Atlas CLI, run the following command:

```
atlas setup [options]
```

Add Your Connection IP Address to Your IP Access List

An IP is a unique numeric identifier for a device connecting to a network. In Atlas, you can only connect to a cluster from a trusted IP address. Within Atlas, you can create a list of trusted IP addresses, referred to as a IP access list, that can be used to connect to your cluster and access your data.**Procedure:**

You must add your IP address to the IP access list before you can connect to your cluster. To add your IP address to the IP access list:

To create an IP access list for your project using the Atlas CLI, run the following command:

```
atlas accessLists create [entry] [options]
```

Create a Database User for Your Cluster

You must create a database user to access your cluster. For security purposes, Atlas requires clients to authenticate as MongoDB database users to access clusters.

Database users are separate from Atlas users:

- Database users can access databases hosted in Atlas.
- Atlas users can log in to Atlas but do not have access to MongoDB databases.
- To create a database user for your project using the Atlas CLI, run the following command:

```
atlas dbusers create [builtInRole]... [options]
```

Connect to Your Cluster

You can connect to your cluster in a variety of ways. In this tutorial, you use one of the following methods:

- The MongoDB Shell, an interactive command line interface to MongoDB. You can use `mongosh` to insert and interact with data on your Atlas cluster.
- MongoDB Compass, a GUI for your MongoDB data. You can use Compass to explore, modify, and visualize your data.
- A MongoDB driver to communicate with your MongoDB database programmatically. To see all supported languages, refer to the MongoDB Driver documentation.

Required Access

To connect to a cluster, you must be a database user.

Prerequisites

Before you start, verify that you have:

- An Atlas account.
- An organization with a project.
- An active cluster created in your account.
- An IP address added to your IP access list.
- A database user on your cluster.

NOTE

You must have a database user set up on your cluster to access your deployment. For security purposes, Atlas requires clients to authenticate as database users to access clusters.

- A terminal
- A text editor
- npm
- Node.js

To install the Node.js driver, run the following command at a terminal prompt:

```
npm install mongodb--save
```

Connect to Your Atlas Cluster

In this section, you get your cluster's connection string from the Atlas UI and connect to your cluster by using your preferred connection method.

To learn about all supported methods, see Connect to Your Database Deployment. For additional driver examples, see Connect via Your Application.

1

In the Atlas UI, select your Database Deployment.

a. Click **Database** in the top-left corner of Atlas.
b. In the **Database Deployments** view, click **Connect** for the database deployment to which you want to connect.

2

Click *Choose a connection method*.

3

Click *Connect your application*.

4

Select `Node.js` from the *Driver* dropdown.

Select your version of the driver from the dropdown. The connection string displays.

5

Copy the provided connection string.

6

Configure the provided connection string.

Replace `<password>` with the password specified when you created your database user.

NOTE

If your passwords, database names, or connection strings contain reserved URI characters, you must escape the characters. For example, if your password is `@bc123`, you must escape the `@` character when specifying the password in the connection string, such as `%40bc123`. To learn more, see Special Characters in Connection String Password.

7

Connect with the sample application.

The following sample application connects to your Atlas cluster with your connection string and returns a confirmation message. To test the sample application, copy the following code into a file called `connect.js`.

11b)Write MongoDB queries to perform CRUD operations on document using insert(), find(), update(), remove()

Answer:

CRUD operations describe the conventions of a user-interface that let users view, search, and modify parts of the database.

When it comes to the individual CRUD operations:

- The **Create operation** is used to insert new documents in the MongoDB database.
- The **Read operation**is used to query a document in the database.
- The **Update operation**is used to modify existing documents in the database.
- The **Delete operation**is used to remove documents in the database.

Create Operations

- For MongoDB CRUD, if the specified collection doesn't exist, the create operation will create the collection when it's executed.

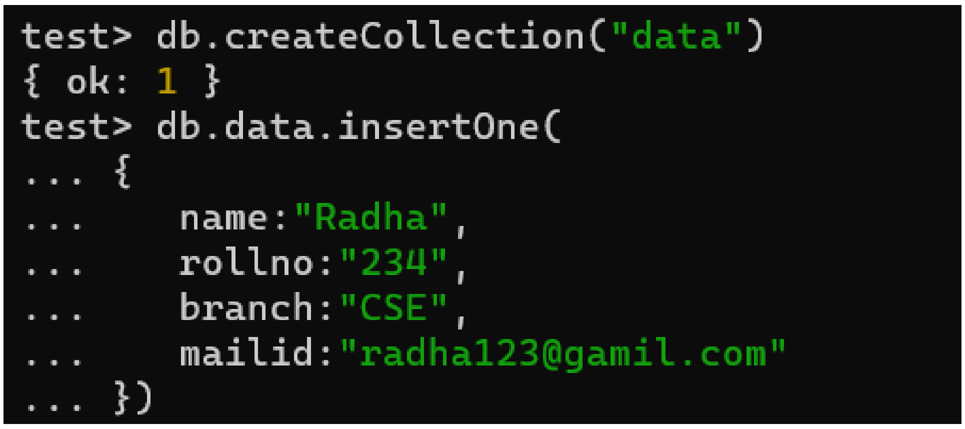
MongoDB provides two different create operations that you can use to insert documents in collection:

- **insertOne()**
- **insertMany()**

insertOne(): As the namesake, insertOne() allows you to insert one document into the collection.

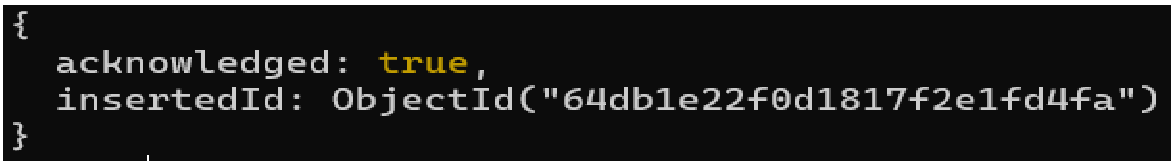
Syntax:db.collection_name.insertOne(document)

Ex:



```
test> db.createCollection("data")
{ ok: 1 }
test> db.data.insertOne(
... {
...     name:"Radha",
...     rollno:"234",
...     branch:"CSE",
...     mailid:"radha123@gamil.com"
... })
```

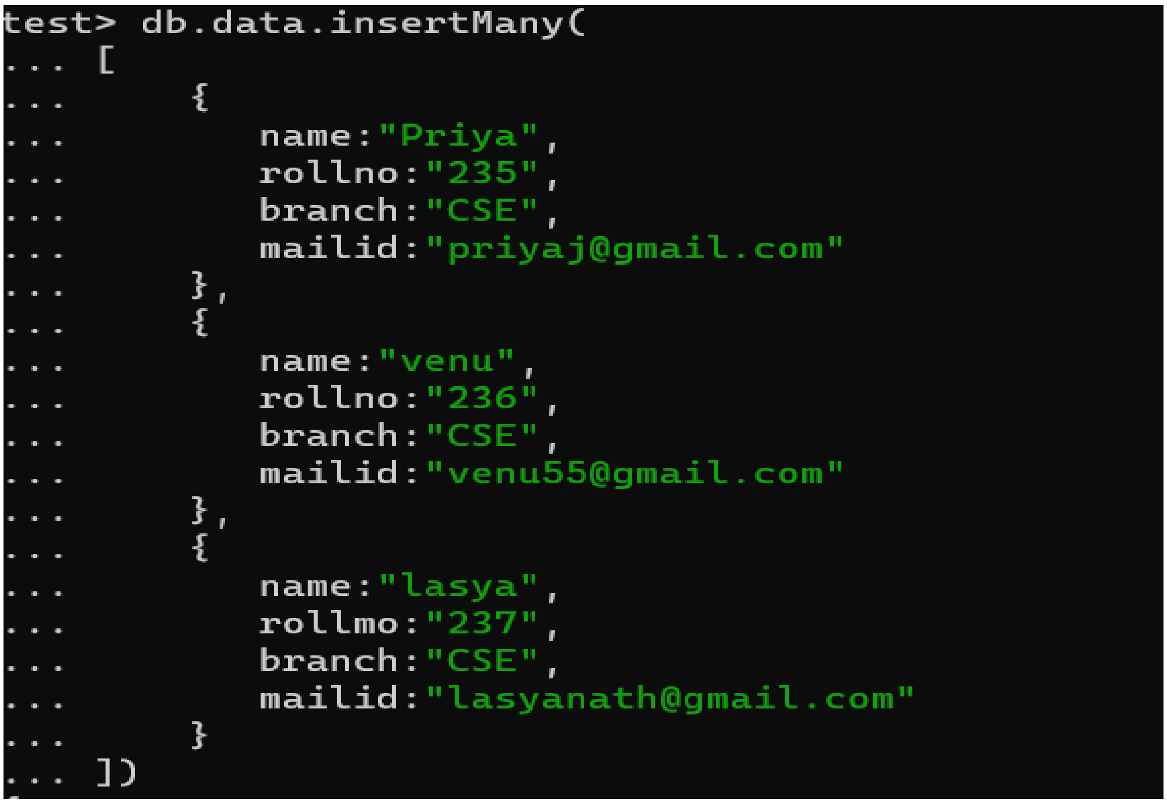
Output:



```
{
  acknowledged: true,
  insertedId: ObjectId("64db1e22f0d1817f2e1fd4fa")
}
```

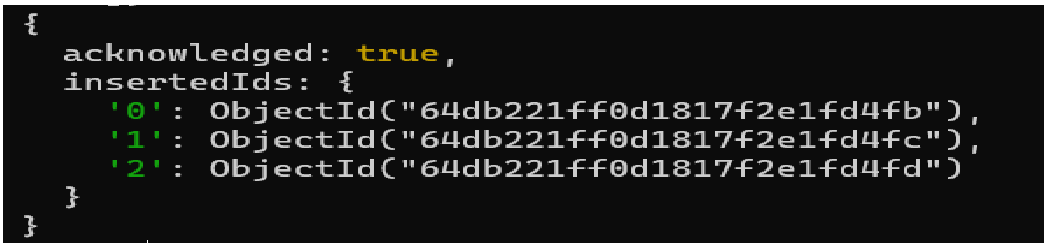
insertMany(): You can insert multiple documents using the insertMany() method. To this method you need to pass an array of documents.

Ex:



```
test> db.data.insertMany(
... [
...      {
...          name:"Priya",
...          rollno:"235",
...          branch:"CSE",
...          mailid:"priyaj@gmail.com"
...      },
...      {
...          name:"venu",
...          rollno:"236",
...          branch:"CSE",
...          mailid:"venu55@gmail.com"
...      },
...      {
...          name:"lasya",
...          rollmo:"237",
...          branch:"CSE",
...          mailid:"lasyanath@gmail.com"
...      }
... ])
```

Output:



```
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("64db221ff0d1817f2e1fd4fb"),
    '1': ObjectId("64db221ff0d1817f2e1fd4fc"),
    '2': ObjectId("64db221ff0d1817f2e1fd4fd")
  }
}
```

Read Operations

The read operations allow you to supply special query filters and criteria that let you specify which documents you want.

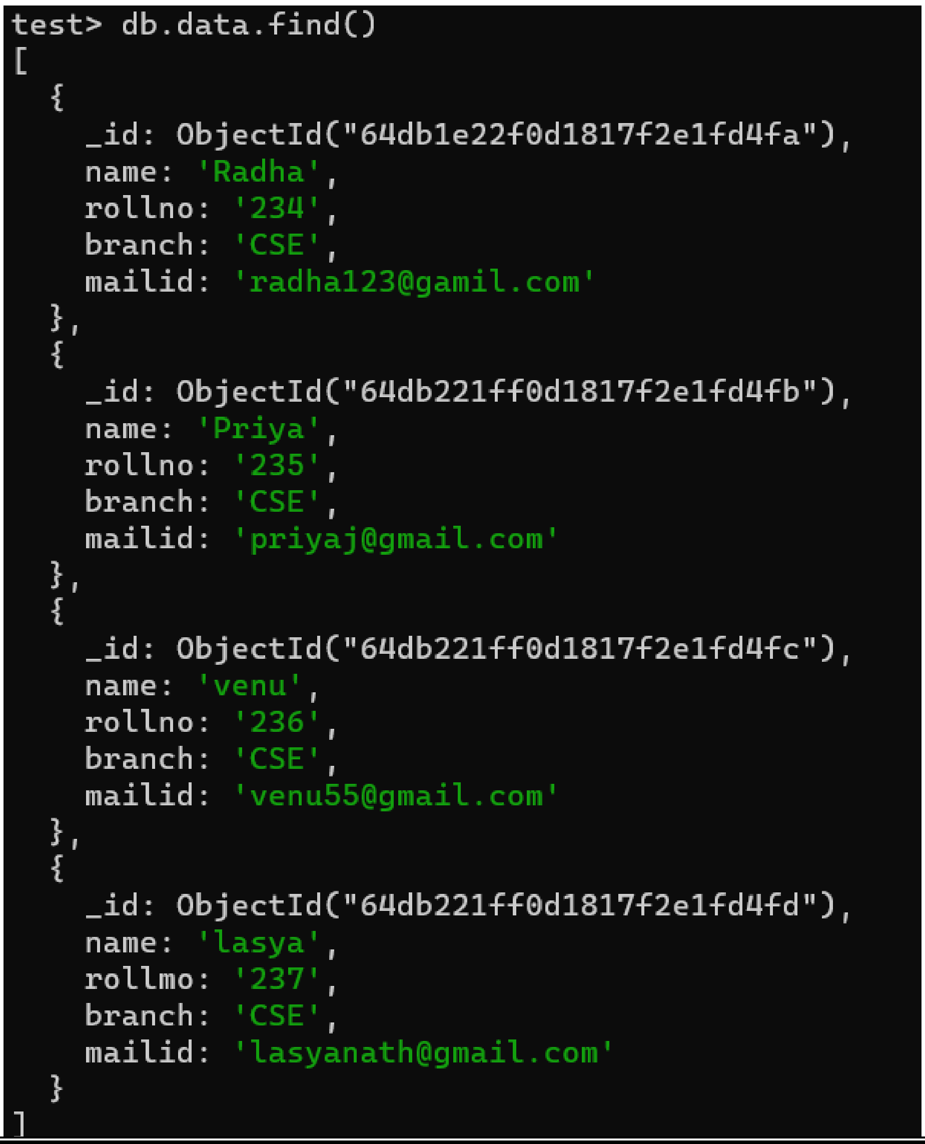
MongoDB has two methods of reading documents from a collection:

- **find()**
- **findOne()**

find(): In order to get all the documents from a collection, we can simply use the find() method on our chosen collection.

Syntax:db.collection_name.find()

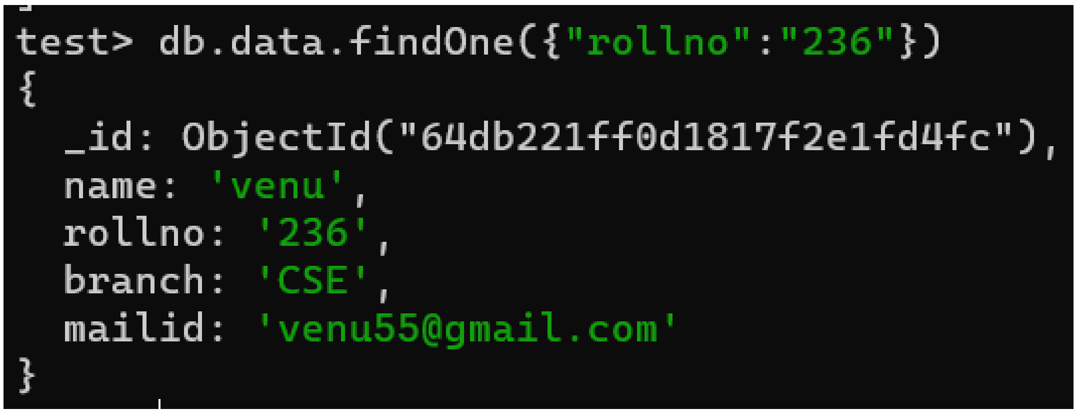
Ex:



```
test> db.data.find()
[
  {
    _id: ObjectId("64db1e22f0d1817f2e1fd4fa"),
    name: 'Radha',
    rollno: '234',
    branch: 'CSE',
    mailid: 'radha123@gamil.com'
  },
  {
    _id: ObjectId("64db221ff0d1817f2e1fd4fb"),
    name: 'Priya',
    rollno: '235',
    branch: 'CSE',
    mailid: 'priyaj@gmail.com'
  },
  {
    _id: ObjectId("64db221ff0d1817f2e1fd4fc"),
    name: 'venu',
    rollno: '236',
    branch: 'CSE',
    mailid: 'venu55@gmail.com'
  },
  {
    _id: ObjectId("64db221ff0d1817f2e1fd4fd"),
    name: 'lasya',
    rollmo: '237',
    branch: 'CSE',
    mailid: 'lasyanath@gmail.com'
  }
]
```

findOne(): In order to get one document that satisfies the search criteria, we can simply use the *findOne()* method on our chosen collection. If multiple documents satisfy the query, this method returns the first document according to the natural order which reflects the order of documents on the disk.

Ex:



```
test> db.data.findOne({"rollno":"236"})
{
  _id: ObjectId("64db221ff0d1817f2e1fd4fc"),
  name: 'venu',
  rollno: '236',
  branch: 'CSE',
  mailid: 'venu55@gmail.com'
}
```

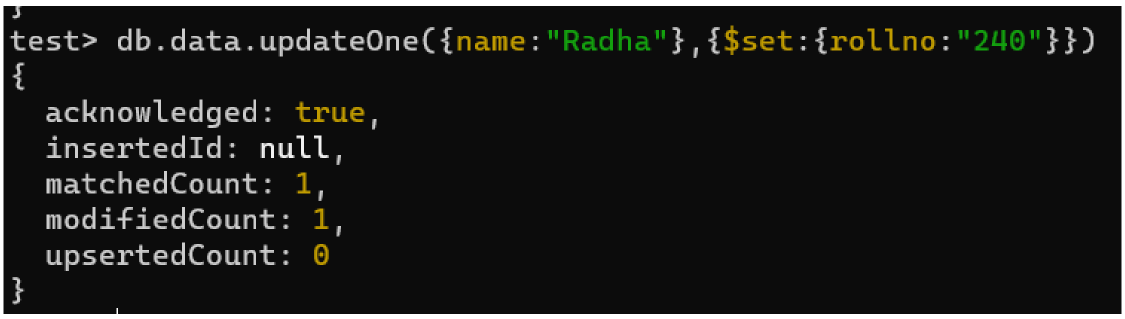
Update Operations

Like create operations, update operations operate on a single collection, and they are atomic at a single document level. An update operation takes filters and criteria to select the documents you want to update.

For MongoDB CRUD, there are three different methods of updating documents:

- **.updateOne()**
- **updateMany()**
- **replaceOne()**

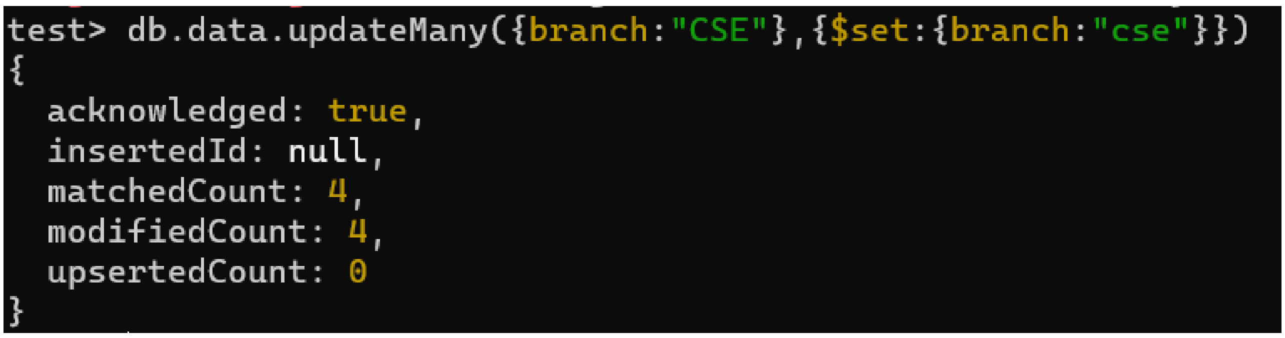
UpdateOne(): We can update a currently existing record and change a single document with an update operation.



```
test> db.data.updateOne({name:"Radha"},{$set:{rollno:"240"}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

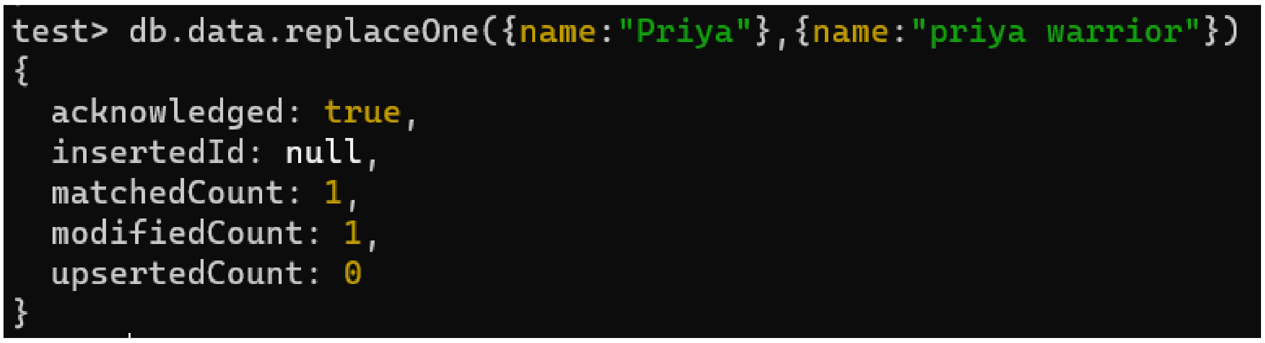
UpdateMany():updateMany() allows us to update multiple items by passing in a list of items, just as we did when inserting multiple items.

Ex:



```
test> db.data.updateMany({branch:"CSE"},{$set:{branch:"cse"}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 4,
  modifiedCount: 4,
  upsertedCount: 0
}
```

replaceOne(): The replaceOne() method is used to replace a single document in the specified collection.



```
test> db.data.replaceOne({name:"Priya"},{name:"priya warrior"})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

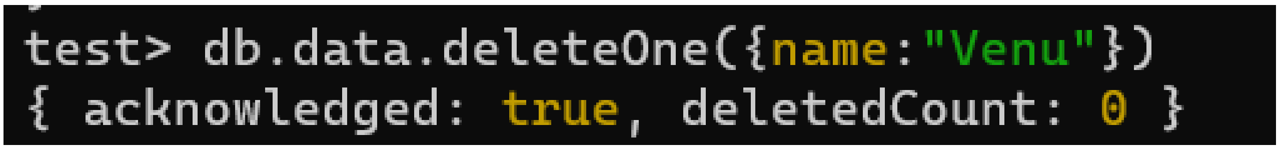
Delete Operations

Delete operations operate on a single collection, like update and create operations. Delete operations are also atomic for a single document.

MongoDB has two different methods of deleting records from a collection:

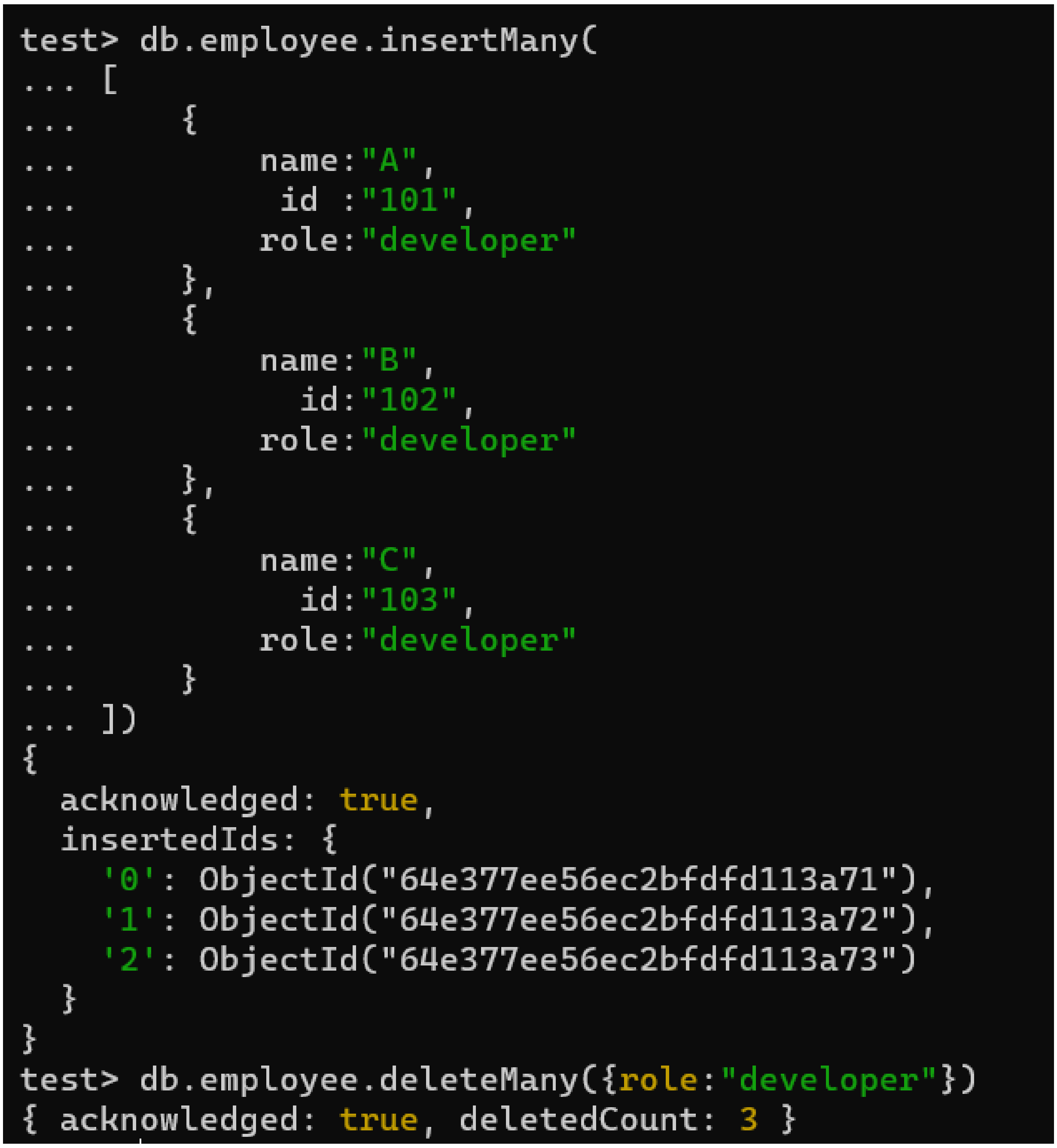
- deleteOne()
- deleteMany()

deleteOne(): *deleteOne()* is used to remove a document from a specified collection on the MongoDB server.



```
test> db.data.deleteOne({name:"Venu"})
{ acknowledged: true, deletedCount: 0 }
```

deleteMany(): deleteMany() is a method used to delete multiple documents from a desired collection with a single delete operation



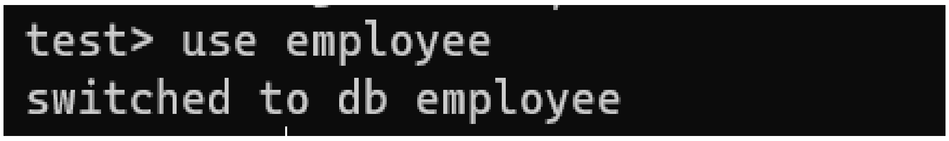
```
test> db.employee.insertMany(
... [
...     {
...             name:"A",
...              id :"101",
...             role:"developer"
...     },
...     {
...             name:"B",
...               id:"102",
...             role:"developer"
...     },
...     {
...             name:"C",
...               id:"103",
...             role:"developer"
...     }
... ])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("64e377ee56ec2bfdfd113a71"),
    '1': ObjectId("64e377ee56ec2bfdfd113a72"),
    '2': ObjectId("64e377ee56ec2bfdfd113a73")
  }
}
test> db.employee.deleteMany({role:"developer"})
{ acknowledged: true, deletedCount: 3 }
```

12a) Write MongoDB queries to Create and drop databases and collections.

Create database:

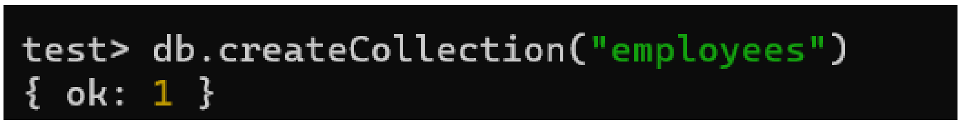
The MongoDB database is a container for collections and it can store one or more collections.

In MongoDB, we can create a database using the use command. As shown in the below image.



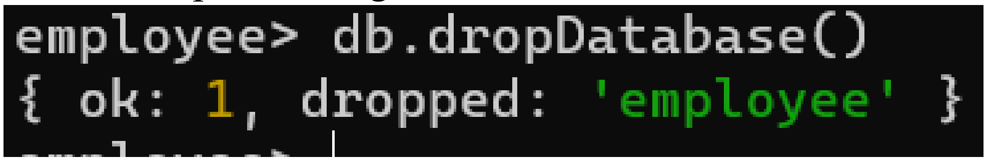
```
test> use employee
switched to db employee
```

Create collection: In MongoDB, a new collection is created when we add one or more documents to it.



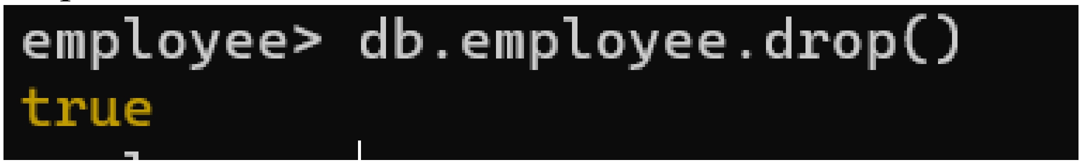
```
test> db.createCollection("employees")
{ ok: 1 }
```

The dropDatabase(): MongoDB **db.dropDatabase()** command is used to drop a existing database.



```
employee> db.dropDatabase()
{ ok: 1, dropped: 'employee' }
```

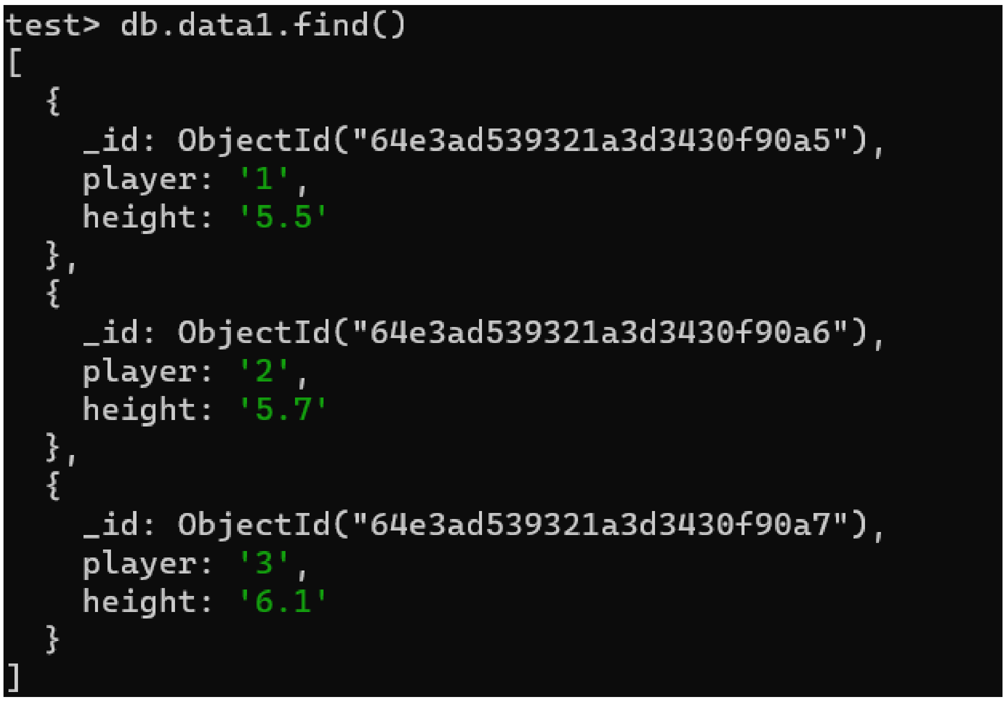
The drop() Method: MongoDB's **db.collection.drop()** is used to drop a collection from the database.



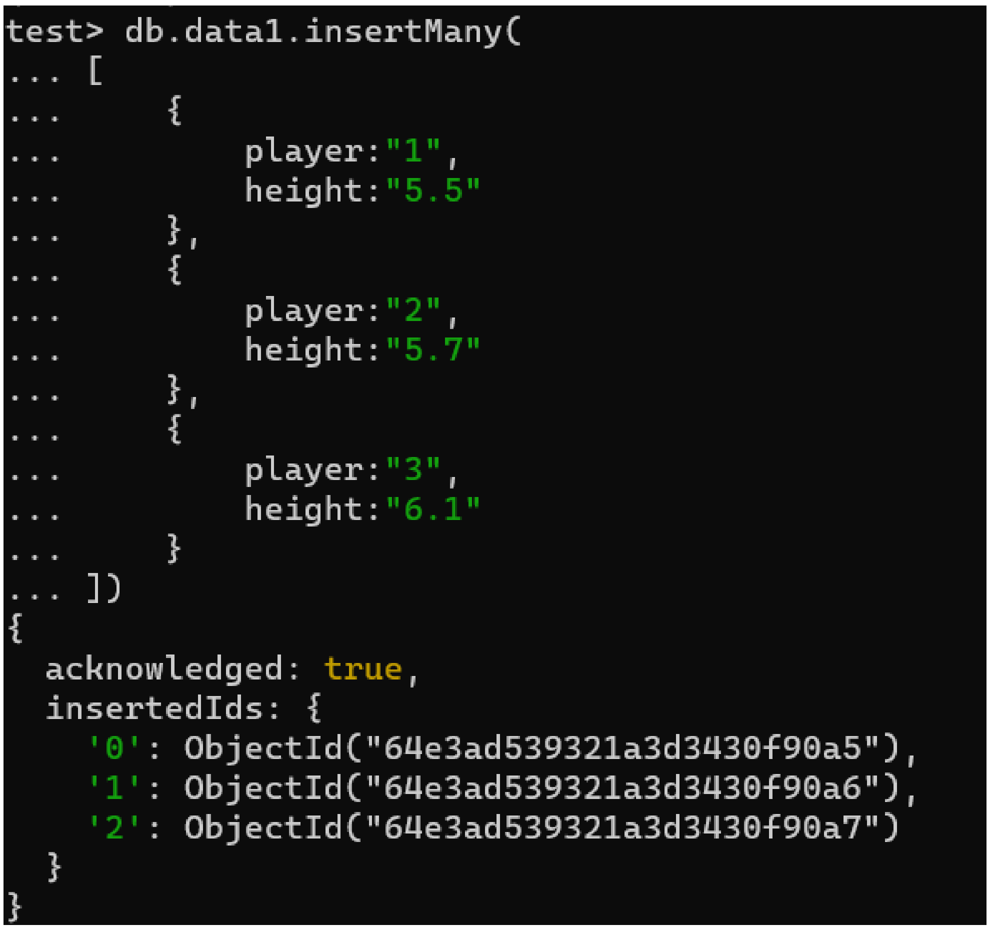
```
employee> db.employee.drop()
true
```

12b)write MongoDB queries to work with records using find(),limit(),sort(),createIndex(),aggregate()

Find method:find() method will display all the documents in a non-structured way.

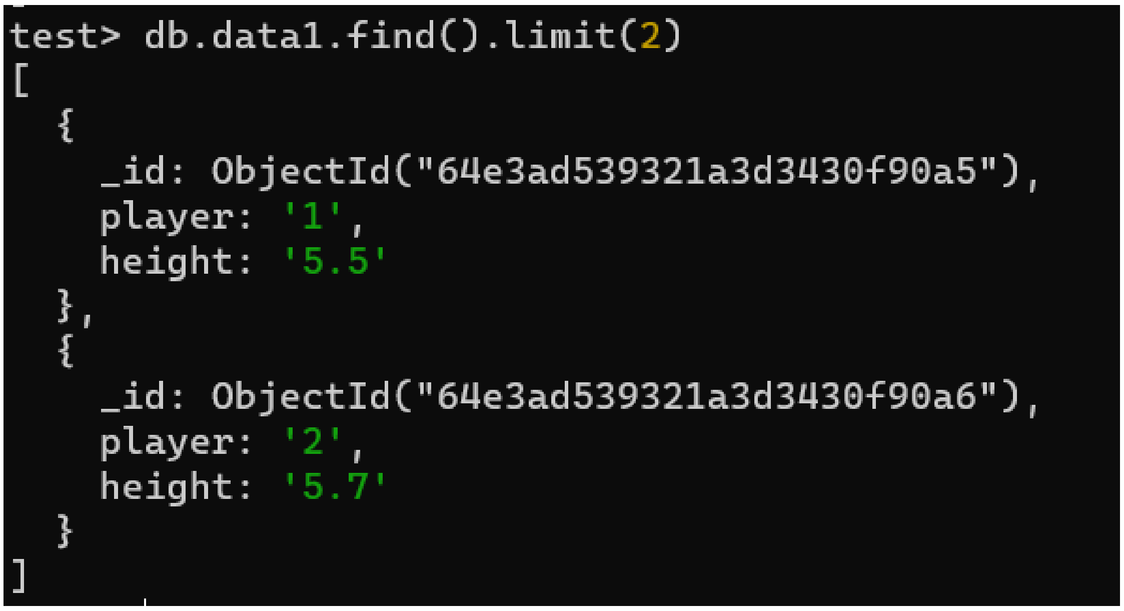


```
test> db.data1.find()
[
  {
    _id: ObjectId("64e3ad539321a3d3430f90a5"),
    player: '1',
    height: '5.5'
  },
  {
    _id: ObjectId("64e3ad539321a3d3430f90a6"),
    player: '2',
    height: '5.7'
  },
  {
    _id: ObjectId("64e3ad539321a3d3430f90a7"),
    player: '3',
    height: '6.1'
  }
]
```



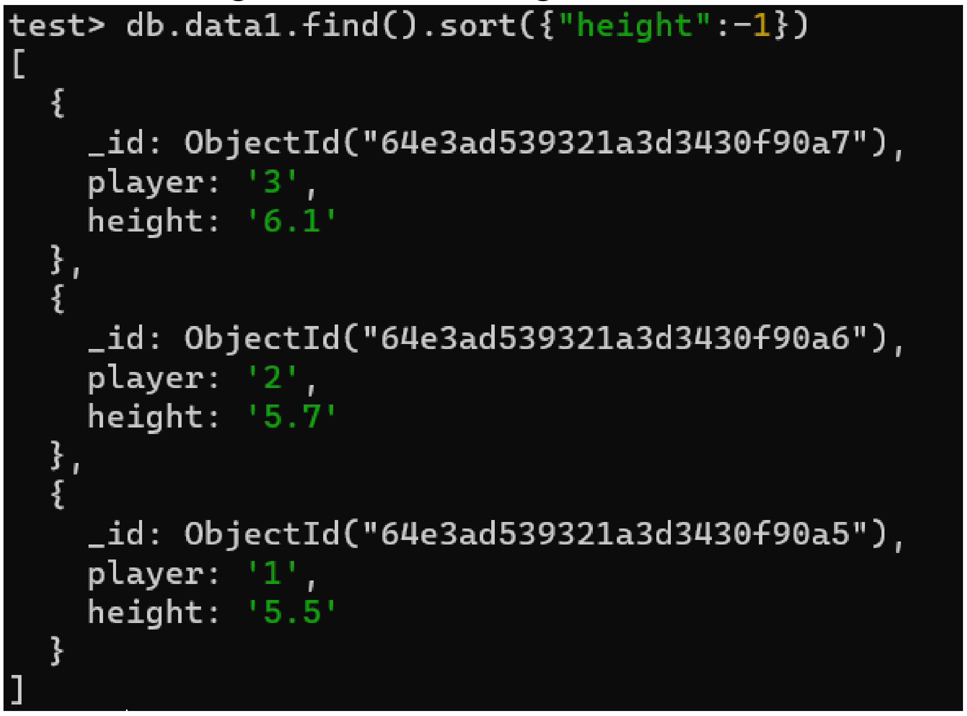
```
test> db.data1.insertMany(
... [
...     {
...         player:"1",
...         height:"5.5"
...     },
...     {
...         player:"2",
...         height:"5.7"
...     },
...     {
...         player:"3",
...         height:"6.1"
...     }
... ])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("64e3ad539321a3d3430f90a5"),
    '1': ObjectId("64e3ad539321a3d3430f90a6"),
    '2': ObjectId("64e3ad539321a3d3430f90a7")
  }
}
```

The Limit() Method: To limit the records in MongoDB, you need to use **limit()** method. The method accepts one number type argument, which is the number of documents that you want to be displayed.



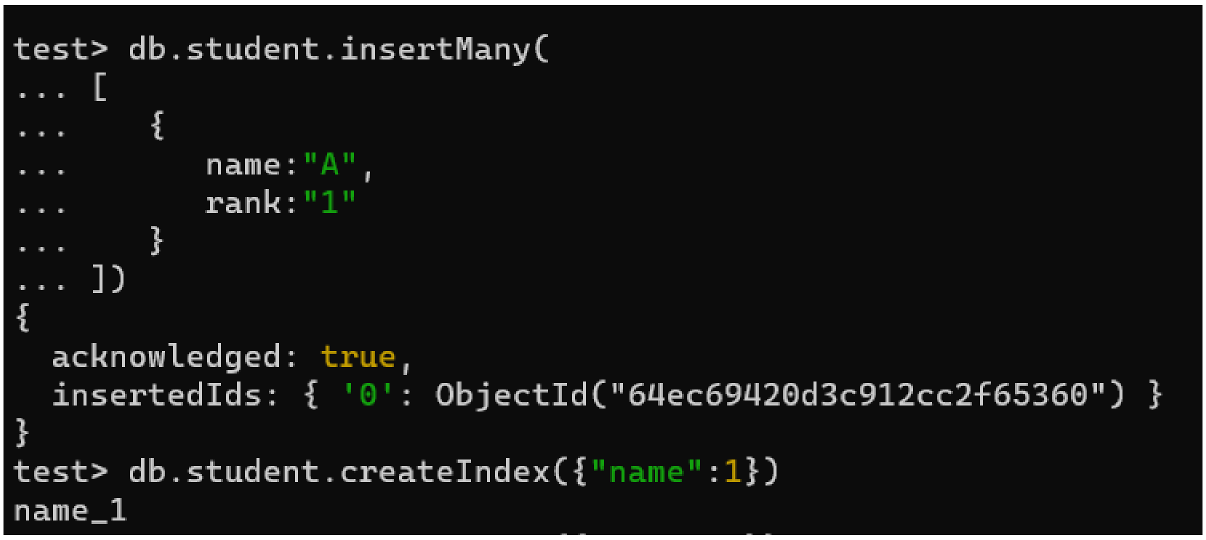
```
test> db.data1.find().limit(2)
[
  {
    _id: ObjectId("64e3ad539321a3d3430f90a5"),
    player: '1',
    height: '5.5'
  },
  {
    _id: ObjectId("64e3ad539321a3d3430f90a6"),
    player: '2',
    height: '5.7'
  }
]
```

The sort() Method : To sort documents in MongoDB, you need to use **sort()** method. The method accepts a document containing a list of fields along with their sorting order.



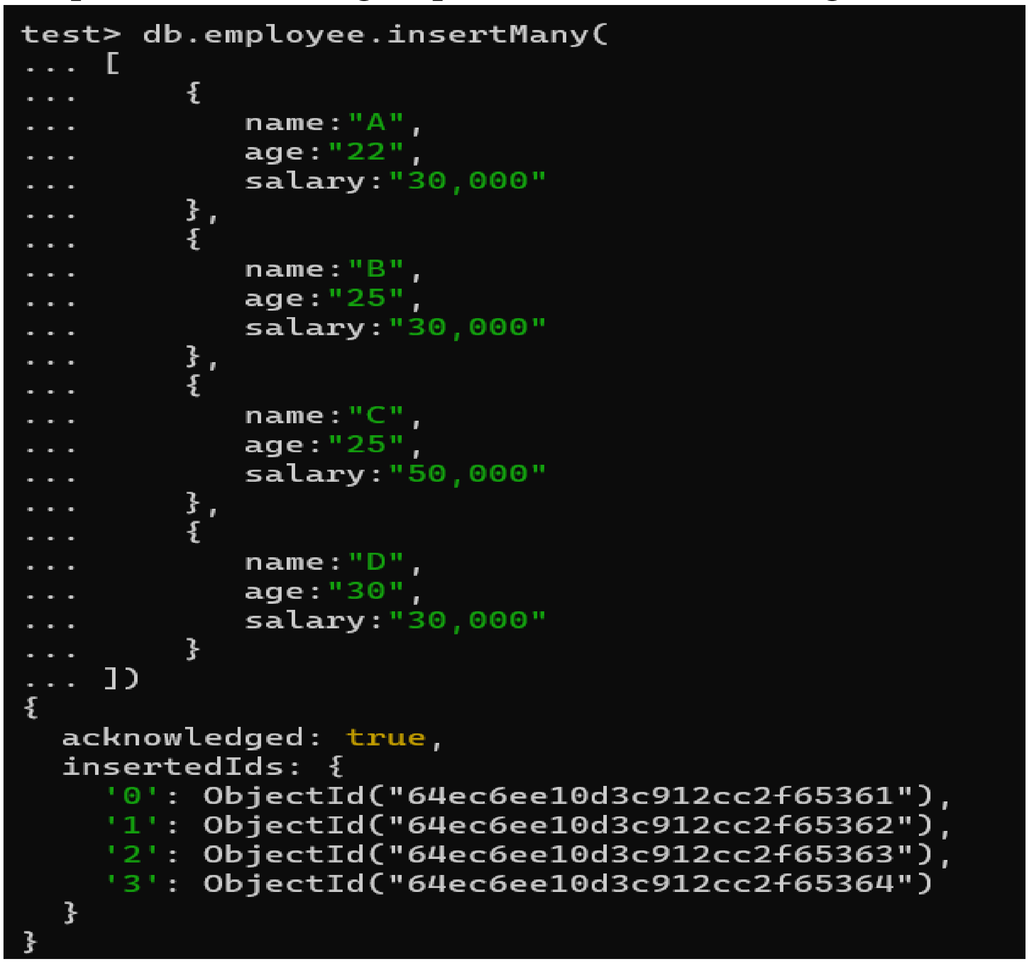
```
test> db.data1.find().sort({"height":-1})
[
  {
    _id: ObjectId("64e3ad539321a3d3430f90a7"),
    player: '3',
    height: '6.1'
  },
  {
    _id: ObjectId("64e3ad539321a3d3430f90a6"),
    player: '2',
    height: '5.7'
  },
  {
    _id: ObjectId("64e3ad539321a3d3430f90a5"),
    player: '1',
    height: '5.5'
  }
]
```

CreateIndex method: In MongoDB, indexes are special data structures that store some information related to the documents such that it becomes easy for MongoDB to find the right data file. The indexes are ordered by the value of the field specified in the index. So, MongoDB provides a **createIndex()** method to create one or more indexes on collections.

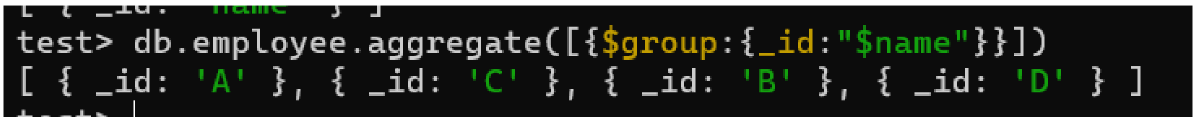


```
test> db.student.insertMany(
... [
...     {
...         name:"A",
...         rank:"1"
...     }
... ])
{
  acknowledged: true,
  insertedIds: { '0': ObjectId("64ec69420d3c912cc2f65360") }
}
test> db.student.createIndex({"name":1})
name_1
```

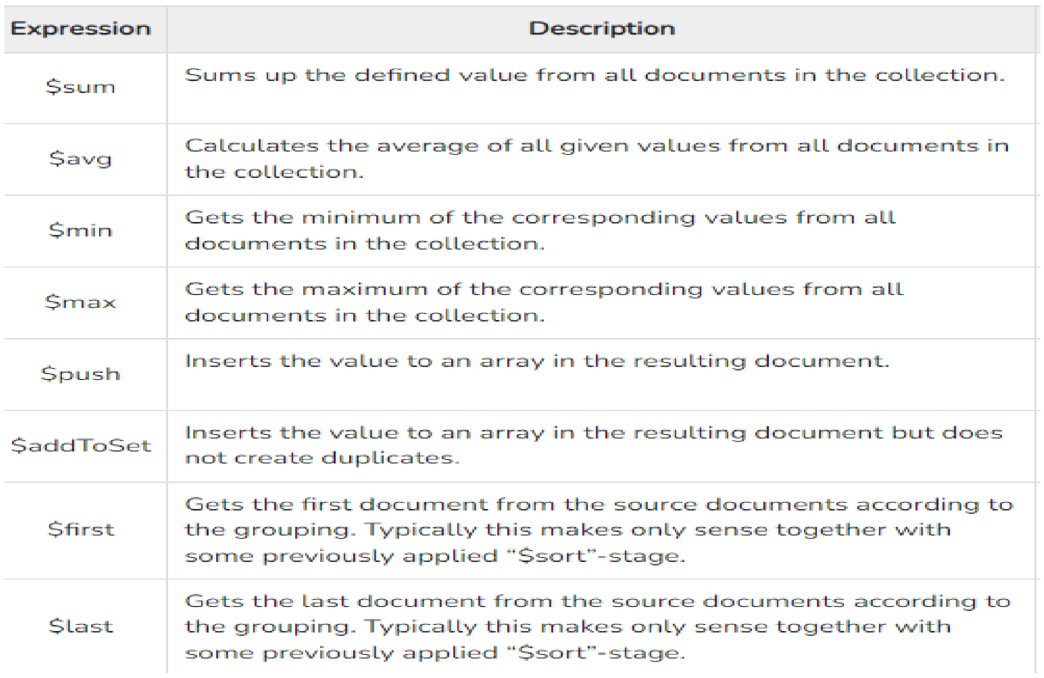
The aggregate() Method: Aggregations operations process data records and return computed results. Aggregation operations group values from multiple documents together, and can perform a variety of operations on the grouped data to return a single result.



```
test> db.employee.insertMany(
... [
...     {
...        name:"A",
...        age:"22",
...        salary:"30,000"
...     },
...     {
...        name:"B",
...        age:"25",
...        salary:"30,000"
...     },
...     {
...        name:"C",
...        age:"25",
...        salary:"50,000"
...     },
...     {
...        name:"D",
...        age:"30",
...        salary:"30,000"
...     }
... ])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("64ec6ee10d3c912cc2f65361"),
    '1': ObjectId("64ec6ee10d3c912cc2f65362"),
    '2': ObjectId("64ec6ee10d3c912cc2f65363"),
    '3': ObjectId("64ec6ee10d3c912cc2f65364")
  }
}
```



```
test> db.employee.aggregate([{$group:{_id:"$name"}}])
[ { _id: 'A' }, { _id: 'C' }, { _id: 'B' }, { _id: 'D' } ]
```



Expression	Description
$sum	Sums up the defined value from all documents in the collection.
$avg	Calculates the average of all given values from all documents in the collection.
$min	Gets the minimum of the corresponding values from all documents in the collection.
$max	Gets the maximum of the corresponding values from all documents in the collection.
$push	Inserts the value to an array in the resulting document.
$addToSet	Inserts the value to an array in the resulting document but does not create duplicates.
$first	Gets the first document from the source documents according to the grouping. Typically this makes only sense together with some previously applied "$sort"-stage.
$last	Gets the last document from the source documents according to the grouping. Typically this makes only sense together with some previously applied "$sort"-stage.